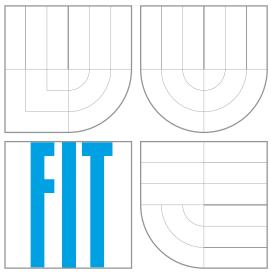
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO AUTOMATIZOVANOU OBSLUHU SUPERPOČÍTAČE

SYSTEM FOR SUPERCOMPUTER AUTOMATION OPERATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER STREČANSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2016

Zadání bakalářské práce/17431/2015/xstrec05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Strečanský Peter**
Obor: Informační technologie
Téma: **Systém pro automatizovanou obsluhu superpočítače**
System for Supercomputer Automation Operation
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s aktuálními prostředky pro obsluhu superpočítačů. Zaměřte se především na systém modulárního softwaru a plánovače OpenPBS.
2. Osvojte si práci se balíčkem FabSim určeným pro automatizovanou obsluhu superpočítače. Zaměřte se na automatickou instalaci knihoven, kompilaci a testování kódu, spouštění uživatelských úloh a přenos souborů.
3. Navrhněte způsob integrace balíčku FabSim do různých simulačních balíčků.
4. Navrženou koncepci realizujte a otestujte na několika superpočítačích.
5. Vytvořte podrobnou uživatelskou dokumentaci pro rychlé osvojení práce se superpočítačem a integraci balíčku FabSim.
6. Zhodnoťte dosažené výsledky a přínos pro běžného uživatele.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Jaroš Jiří, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Cieľom tejto práce je rozšíriť existujúci program FabSim o modul, ktorý umožní automatizovanú obsluhu superpočítačov, predovšetkým prácu s plánovačom OpenPBS. Modul bol vytvorený v programovacom jazyku Python za použitia balíčka Fabric. Skripty, ktoré sa pomocou OpenPBS spúšťajú, sú uložené vo forme predlôh a pred samotným prenosom na cluster a spustením sa dynamicky upravia na základe preferencií užívateľa. Vytvorené riešenie tak poskytuje komplexnú sadu metód, ktoré umožňujú plnohodnotnú obsluhu superpočítačov, integráciu s Gitom a spravovanie dát nachádzajúcich sa na superpočítačoch. Hlavným prínosom tejto práce je jednoduchšie riadenie a úspora času spojená s obsluhou superpočítačov.

Abstract

The main goal of this thesis is to extend already existing software FabSim by a module, which allows automated supercomputer operation, especially with OpenPBS scheduler. This module was implemented with Python programming language, using Fabric module as its backbone. The scripts, which are executed with OpenPBS are stored as the templates. These templates are dynamically modified to suit users needs. This solution provides a complex set of methods, which allows full–featured operation of supercomputers, integration with git and data management on clusters. The module saves time and makes working with supercomputers much easier.

Kľúčové slová

superpočítač, cluster, Fabric, Python, softvérové inžinierstvo, OpenPBS

Keywords

supercomputer, cluster, Fabric, Python, software engineering, OpenPBS

Citácia

STREČANSKÝ, Peter. *Systém pro automatizovanou obsluhu superpočítače*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroš Jiří.

Systém pro automatizovanou obsluhu superpočítače

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jiřího Jaroše, Ph.D. Uviedol som všetky literáre pramene a publikácie, z ktorých som čerpal.

.....
Peter Strečanský
18. mája 2016

Poďakovanie

Táto práca vznikla za podpory Ministerstva školství, mládeže a tělovýchovy a projektu „IT4Innovations národní superpočítačové centrum – LM2015070“.

© Peter Strečanský, 2016.
Táto práca vznikla ako školské dielo na FIT VUT v Brně. Práca je chránená autorským zákonom a jej využitie bez poskytnutia oprávnenia autorom je nezákonné, s výnimkou zákonne definovaných prípadov.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je rozšíriť program určený na automatizovanú obsluhu vzdialených zariadení tak, aby bolo pomocou neho možné plohodnotne riadiť superpočítače Anselm a Salomon. Obsluhu programu musia byť schopní zvládnuť aj ľudia, ktorí nemajú vzdelanie v oblasti informatiky a ich počítačová gramotnosť je na priemernej úrovni.

Celý proces vytvárania nového modulu je rozdelený do niekoľkých častí. Aby bolo možné vôbec s implementáciou začať, je potrebné najprv riadne porozumieť problematike superpočítačov. Ich história, architektúra a vývoj sú spolu s technickou špecifikáciou superpočítačov Anselm a Salomon rozobraté v kapitole 2. Následne je nutné pochopiť princíp akutálnej verzie programu FabSim, jeho špecifikám a nedostatkom. Tým je venovaná kapitola 3. V kapitole 4 si predstavíme vybrané metódy so softvérového inžinierstva, vďaka ktorým okúsime postup vývoja softvéru podobný tomu v praxi a ktorý tento vývoj o poznanie zjednoduší.

Praktická časť práce je venovaná samotnej implementácii programu. Jej postup, špecifiká a jednotlivé programátorské praktiky sú zhrnuté v kapitole 5. Aby bol výsledný program spoľahlivý a okamžite pripravený na prevádzku, je nutné ho riadne otestovať. Vybrané typy testovania, postup a získaná spätná väzba sú rozobraté v kapitole 6. Na záver, kapitola 7 predstavuje kompletnú užívateľskú príručku, ktorá obsahuje všetky pridané metódy a pokrýva plnú funkcionalitu vytvoreného modulu.

Zámerom je, aby výsledný produkt, teda rozšírená verzia programu FabSim, bola nenáročná na obsluhu s plnohodnotnou užívateľskou dokumentáciou. Takýto program umožní zrýchliť a zjednodušiť prácu ľudí v tíme, v ktorom každá ušetrená minúta pri používaní superpočítačov znamená nielen značnú úsporu financií, ale aj možnosť efektívne využiť ušetrený čas na výskum technológie, ktorá raz môže zachraňovať životy či, pomáhať pri hľadaní nerastných surovín.

Kapitola 2

Superpočítače

Superpočítačom sa nazýva taký počítačový systém[1], ktorého výkon je na alebo blízko súčasnej hranice najvyššej prevádzkovej rýchlosti počítačov [12]. Presná hranica, odkedy možno považovať konkrétny počítačový systém za superpočítač, však určená nie je, líši sa v závislosti od zdroja z ktorého čerpáme. Pojem superpočítač bol spopularizovaný v roku 1964, keď bol vyrobený superpočítač CDC 6600, ktorý navrhol Seymour Cray a bol vyrobený spoločnosťou Control Data Corporation. V súvislosti s pojmom superpočítač sa často používa aj pojem cluster či HPC (High - performance computing) systém.

2.1 Architektúra

Superpočítače sa od klasických izbových počítačov odlišujú okrem výkonu najmä vzhľadom, resp. konštrukciou. Keďže sa jedná o stroje, ktoré zaberajú plochu celých miestností, musia byť jeho jednotlivé súčasti riadne organizované. Typicky sa superpočítač skladá z týchto súčastí:

- Šasi (chassis) - základná konštrukcia, v ktorej je umiestnený jeden alebo viac uzlov
- Uzol (node) - obvykle plošný spoj, vyrábaný s niekoľkými prázdnymi socketmi
- Socket - Dá sa doň zapojiť jeden procesor
- Procesor - centrálna procesová jednotka, skladá sa z viaceých jadier
- Jadrá (cores) - vykonávajú samotné operácie

2.2 Meranie výkonu

Výkon superpočítačov sa meria v jednotkách FLOPS, čo je skratka pre Floating Point Operations per Second, teda počet floating point operácii (najčastejšie sčítanie a násobenie) za sekundu. Teoretický výkon systému v GFLOPS[2] je možné vypočítať pomocou rovnice 2.1 [4]:

$$GFLOPS = chassis \times \frac{nodes}{chassis} \times \frac{sockets}{node} \times \frac{cores}{socket} \times \frac{GHz}{core} \times \frac{FLOPs}{cycle} \tag{2.1}$$

[1] Počítačový systém sa skladá z hardware, systémového a aplikačného software
[2] $10^9 FLOPS$

V súčasnosti väčšina mikroprocesorov dosahuje výkon v desiatkach až stovkách GFLOPS. Pri superpočítačoch sa jedná o výkon rádovo v desaťtisícoch TFLOPS[3][18]. Zároveň treba zopakovať, že použitím danej rovnice sa získa teoretický výkon systému. Skutočný výkon sa vyjadruje na základe rôznych benchmarkov ako napríklad LINPACK.

LINPACK bol vymyslený Jackom Dongarrom a začal sa prvý krát používať v roku 1974. Princípom tohto benchmarku je výpočet pomocou n lineárnych rovníc typu $Ax = b$. Postupne sa používali verzie LINPACK 100, LINPACK 1000 a na paralelné výpočty sa používa HPLINPACK. Najaktuálnejšie verzie sa používajú na zostavenie rebríčka TOP500.

2.3 Použitie

Už zo samotného názvu vyplýva, že superpočítače sa nepoužívajú takým spôsobom ako väčšina iných počítačov. Obrovský výpočtový výkon ich predurčuje na riešenie tých najzložitejších úloh. Vuyžívajú sa v širokom spektre oborov ako napríklad kvantová mechanika, molekulárne modelovania, predpovede počasia [2], predikcia zmeny klímy či simulácie molekulárnej dynamiky [5].

Okrem využitia na vedecké účely sa používajú aj na nevedecké účely. V roku 1997 superpočítač Deep Blue porazil v sérii 6 zápasov šachového veľmajstra Garyho Kasparova a stal sa tak prvým strojom, ktorému sa podarilo Kasparova poraziť [8].

2.4 Superpočítače Anselm a Salomon

Superpočítače Anselm a Salomon, pre ktoré je táto práca primárne určená, sa nachádzajú v Ostrave a patria pod správu národného počítačového centra IT4Innovations[4]. IT4Innovations je súčasťou Vysoké školy báňské - Technické univerzity Ostrava. Na riadení a realizácii centra IT4Innovations sa podieľajú aj Vysoké učení technické v Brně, Ostravská univerzita v Ostravě, Slezská univerzita v Opavě a Ústav geoniky AVČR.

Ako prvý bol v máji roku 2013 uvedený do prevádzky superpočítač Anselm, ktorý bol počas prechodného obdobia umiestnený v provizórnych vonkajších kontajneroch. V júli 2015 bol uvedený do prevádzky aj superpočítač Salomon, ktorý je navyše podľa rebríčka TOP500 48. najvýkonnejším superpočítačom na svete[5]. Od leta 2015 sú navyše obidva superpočítače umiestnené v novom areáli.

V nasledujúcej kapitole sú rozobrané technické detaily týchto superpočítačov a vysvetlený princíp plánovania úloh a modulárneho softvéru. Podrobnú dokumentáciu k obom clustrom je možné nájsť na stránke https://docs.it4i.cz/get-started-with-it4innovations.

2.4.1 Technická špecifikácia

Anselm

Superpočítač Anselm pozostáva celkovo z 209 výpočtových uzlov organizovaných v 13 šasi. Z týchto uzlov je 180 bez akcelerátora, 23 s GPU akcelerátorom NVIDIA Tesla Kepler K20, 4 s MIC akcelerátorom Intel Phi 5110P a dva tzv. „fat“ uzly, ktoré sa používaju pri hybridných paralelných kódoch, ktoré majú veľké pamäťové a/alebo viacvláknové nároky.

[3] $10^{12} FLOPS$

[4] http://www.it4i.cz

[5] http://www.top500.org/list/2015/11/

Obr. 2.1: Superpočítač Salomon [7]

Jednotlivé uzly sú osadené procesormi Intel Sandy Bridge E5-2665 a Intel Sandy Bridge E5-2470. Maximálny výpočtový výkon jedného jadra je 19,2 Gflop/s pre procesory E5-2665, resp. 18.4 Gflop/s pre procesory E5-2470. Každý uzol bez akcelerátora obsahuje 8 x 8 GB pamäte RAM, uzly s GPU a MIC akcelerátormi obsahujú 6 x 16 GB pamäte RAM na uzol a „fat“ uzly sú osadené 16 x 32 GB RAM na uzol.

Na clusteri sa nachádzajú dva hlavné uložné priestory, a to `HOME` a `SCRATCH`. Obidva pamäťové systémy sú realizované ako parelelné Lustre súborové systémy. V súborovom systéme `HOME` má každý užvateľ zriadený vlastný priečinok `/home/username`, kde `username` je login užívateľa. Jednotliví užívatelia sú limitovaní na vlastný, nezdieľaný pamäťový priestor o veľkosti 250 GB [6]. Tento súborový systém primárne slúži na prípravu, hodnotenie, spracovanie a ukladanie dát generovaných aktívnymi projektami.

Každý užívateľ má v súborovm systéme `SCRATCH` svoj súkromný pamäťový priestor o veľkosti 100 TB. Na tomto súborovom systéme by sa mali nachádzať všetky dočasné dáta vznikajúce v priebehu výpočtu a takisto všetky vstupné a výstupné súbory. Súbory, ku ktorým sa nepristúpi viac ako 90 dní sa automaticky zmažú.

Prístup na cluster je realizovaný dvoma prihlasovacími uzlami na adrese `anselm.it4i.cz`. Autentifikácia prebieha pomocou súkromného ssh kľúča. Na clusteri sa taktež nachádza uzol určený pre prenos dát. Prenos dát prebieha pomocou protokolu scp, pričom maximálna rýchlosť prenášania dát je 160 MB/s. Všetky výpočtové a prihlasovacie uzly sú spojené pomocou siete Infiniband QDR a Gigabit Ethernet.

Salomon

Architektúra clusteru Salomon je podobná tej, aká je na superpočítači Anselm. Salomon sa skladá z 1008 výpočtových uzlov. Z tohto počtu je 576 uzlov bez akcelerátora. Každý z týchto uzlov je osadený dvoma procesormi Intel Xeon E5-2680v3. Maximálny výpočtový výkon tohto procesora je 19,2 Gflop/s na jadro. Každý z týchto uzlov obsahuje 8 x 16 GB DDR4 pamäte RAM. Týmito procesormi a RAM pamäťou je vybavených aj ďalších 432 uzlov, ktoré navyše obsahujú dva MIC koprocesory Intel Xeon Phi 7120P. Maximálny výpočtový výkon každého jadra akcelerátora je 18,4 Gflop/s. Každý z týchto 61 – jadrových

[6] Po požiadaní je však možné túto kapacitu zväčšiť

koprocesorov je vybavený 16 GB DDR4 RAM pamäťou. Jeden uzol s názvom *UV 2000* je vybavený štrnástimi 8–jadrovými procesormi Intel Xeon E5-4627v2. Tento uzol obsahuje 3328 GB pamäte RAM a navyše aj GPU NVIDIA GeForce GTX Titan X s jadrom NVIDIA GM200 s 12GB RAM.

Na Salomone sa, takisto ako aj na Anselme, nachádzajú dva zdieľané úložné priestory `HOME` a `SCRATCH`, na ktoré majú prístup všetky prihlasovacie a výpočtové uzly. Súborový systém `HOME` je realizovaný ako viacstupňová NFS disková pamäť. V tomto pracovnom mieste má každý užívateľ zriadený svoj priečinok na adrese `/home/username`, kde `username` je prihlasovacie meno užívateľa. Každý užívateľ má k dispozícii svoj nezdieľaný pamäťový priestor o veľkosti 250 GB. Použitie tohto úložiska je rovnaké ako na clusteri Anselm, teda na prípravu, hodnotenie, spracovanie a ukladanie dát generovaných aktívnymi projektami.

Na súborovom systéme `SCRATCH` sídlia dve časti (workspaces) `WORK` a `TEMP`. Diskový priestor `WORK` by mal obsahovať uložené dáta súvisiace s projektom, či vstupné alebo výstupné súbory. V pracovnom priestore `TEMP` môžu užívatelia vytvárať ľubovoľné priečinky či súbory. Mali by sa tu nachádzať všetky dočasné dáta vznikajúce v priebehu výpočtu, prípadne aj vstupné a výstupné súbory. Súbory, ku ktorým sa nepristúpi po dobu viac ako 90 dní sú automaticky zmazané.

Superpočítač Salomon obsahuje až štyri prihlasovacie uzly. Princíp je totožný s clusterom Anselm, jediným rozdielom je, že na Salomone nie je pre prenos súborov vyhradený žiaden špeciálny uzol. Všetky uzly sú prepojené pomocou siete Infiniband (56 Gbps) a Gigabit Ethernet sieťou.

2.4.2 Exekúcia programov

Na to, aby sa na clusteroch mohol spustiť program, prípadne preniesť väčšia porcia dát (obecne sa všetky tieto úlohy nazývaju anglickým slovom jobs), je potreba alokovať výpočtové prostriedky na určitý čas. Daná úloha sa následne zaradí do fronty a na základe plánovania sa o určitú dobu neskôr úloha vykoná. Princíp alokovania zdrojov, plánovania a exekúcie úloh je na oboch clusteroch rovnaký (až na malé odchýlky v špecifikáciách jednotlivých front, ktoré sú dané rôznym počtom uzlov na clusteroch). Ako príklad preto budeme uvádzať spôsob plánovania a exekúcie úloh na clusteri Anselm.

Alokácia zdrojov

Pred vykonaním príslušnej úlohy si musí užívateľ alokovať určitú časť clustera na dobu potrebnú na dokončenie úlohy. Tieto úlohy sa následne zaradia do fronty špecifikovanej užívateľom. Na clusteri Anselm sa nachádza 7 druhov front, pričom každá z nich má rôzne využitie, počet pridelených uzlov, prioritu a maximány čas alokácie.

Fronta `qexp` je tzv. expresná fronta. Je určená na testovanie a exekúciu iných, krátkych úloh. Maximálny počet uzlov (vrátane uzlov s akcelerátormi, či „fat“ uzlov), ktoré si užívateľ môže alokovať, je 8. Doba exekúcie (walltime) je nastavená na maximálnu dĺžku 1 hodina.

Fronta `qprod` je určená pre štandardný beh úloh, nazýva sa preto aj produkčná fronta. Pomocou tejto fronty je možné alokovať všetky okrem rezervovaných uzlov, maximálna doba exekúcie je 48 hodín.

Fronta `qlong` sa používa na exekúciu úloh, ktoré zaberajú dlhšiu dobu než je maximálna doba exekúcie vo fronte `qprod`. Maximálna doba exekúcie je stanovená na trojnásobok dĺžky vo fronte qprod, teda 144 hodín.

Medzi špecializované fronty patria `qnvidia`, `qmic` a `qfat`. Ako už ich názov napovedá, používajú sa k prístupu na uzly s Nvidia akcelerátormi, MIC koprocesormi či na „fat“ uzly.

Tieto špecializované fronty majú výsokú prioritu, čo znamená že úlohy v týchto frontách sa vykonajú skôr ako v iných frontách.

Poslednou frontou je fronta `qfree`. Výpočtový čas na tejto fronte je zdarma a používa sa najmä pri projektoch, ktorých všetky zdroje už boli alokované či vyčerpané. Na tejto fronte je možné alokovať maximálne 178 uzlov bez akcelerátorov pričom maxmálna doba exekúcie je stanovená na 12 hodín.

Plánovanie úloh

Určenie času, v ktorom sa konkrétna úloha začne vykonávať má na starosti tzv. plánovač. Ten prideluje každej úlohe prioritu, na základe ktorej sa úloha vykoná skôr či neskôr v porovnaní s ostatnými úlohami. Priorita úlohy je daná na základe týchto faktorov (faktory sú zoradené podľa ich dôležitosti):

- Priorita fronty
- „Fairshare“ priorita
- Zvyšný čas daného projektu (Eligible time)

Každá fronta, v ktorej je úloha umiestnená pred vykonaním, má určenú prioritu (ako už bolo spomínané, napríklad špecializované fronty `qnvidia`, `qmic` a `qfat` majú pridelenú veľmi vysokú prioritu). Táto priorita má najväčší vplyv na určenie času začiatku vykonávania úlohy. Čím má fronta vyššiu prioritu, tým je väčšia aj výsledná priorita úlohy.

Princípom „Fairshare“ priority je, aby každý užívateľ mohol využiť približne rovnakú časť výpočtových zdrojov za týždeň. „Fairshare“ priorita sa používa na zoradenie úloh, ktorých fronty majú rovnakú prioritu. Táto priorita sa počíta pre projekt a teda je rovnaká pre všetkých užívateľov, ktorí pod daný projekt spadajú. Projekty, ktoré v poslednej dobe častejšie využívali zdroje clustera majú nižšiu „Fairshare“ prioritu ako projekty s menšou či žiadnou aktivitou. Jej hodnota sa vypočíta podľa vzorca 2.2:

$$MAX\_FAIRSHARE * (1 - \frac{usage_{Project}}{usage_{Total}}) \tag{2.2}$$

kde $MAX\_FAIRSHARE$ má hodnotu $1E6$, $usage_{Project}$ je doba využitia clusteru všetkými členmi daného projektu a $usage_{Total}$ je doba využitia clusteru všetkými projektami. Doba využitia clusteru je daná ako počet alokovaných hodín a jej hodnota je výsledkom súčinu počtu jadier a doby na ktorú boli alokované. Úlohy vo fronte `qexp` sa do tejto hodnoty nerátajú. Táto hodnota je nulovaná, prípadne zmenšená na polovicu pravidelne v intervaloch 168 hodín, teda raz za týždeň.

Eligible time je čas v sekundách, ktorý zostáva do vyčerpania zdrojov daného projektu. Táto hodnota má najmenší vplyv na výslednú prioritu úlohy. Používa na zoradenie úloh s rovnakou prioritou fronty a „Fairshare“ prioritou.

Výsledná priorita úlohy sa vypočíta podľa vzorca 2.3:

$$1000 * queue\_priority + \frac{fairshare\_priority}{1000} + \frac{eligible\_time}{864000} \tag{2.3}$$

2.4.3 Rozsiahle úlohy a úlohy s využitím paralelizmu

Pri vykonávaní náročných programov ako sú napríklad simulácie či paralelné programy, je vhodné rozdeliť jednu úlohu na viac menších podúloh a každej z nich alokovať príslušné

zdroje. Rozdelenie úlohy na čo najväčší počet menších úloh je najefektívnejší spôsob využitia výpočtového výkonu superpočítača. Táto metóda zaisťuje najkratšiu dobu výpočtu, optimalizáciu výkonu superpočítača a v konečnom dôsledku aj nižšie náklady spojené s výpočtom.

Pri príliš veľkom počte úloh však môže dôjsť k zahlteniu systému, z čoho vyplýva neefektívne plánovanie a celková degradácia výkonu. Z tohto dôvodu je počet úloh na užívateľa limitovaný na 100. Ak však užívateľ potrebuje z akéhokoľvek dôvodu naplánovať a vykonať viac úloh, odporúča sa použiť nasledujúce metódy:

Polia úloh

Pole úloh (z anglického Job array) je štruktúra podobná poliam ako ich poznáme z programovania. Táto štruktúra zastrešuje veľké množstvo úloh, ktoré sa nazývajú podúlohy. Všetky podúlohy v jednom poli zdieľajú ten istý jobscript vygenerovaný plánovačom. Každá podúloha má svoj unikátny index, ktorý je uložený v premennej `$PBS_ARRAY_INDEX`, ich identifikátory sa líšia len v indexe a každá podúloha má svoj vlastný stav. Okrem týchto troch atríbutov sú všetky ostatné rovnaké. Kapacita jedného poľa je 1000 podúloh. Polia úloh je odporúčané využívať pri veľkom množstve viacvláknových či viacuzlových úloh.

GNU Parallel

GNU Parallel je nástroj shellu[7], ktorý je určený na paralelné vykonávanie úloh za použitia jedného alebo viacerých počítačov. Využitie tohto nástroja je najužitočnejšie pri vykonávaní viacerých úloh, ktoré na svoje vykonanie potrebujú len jedno jadro procesora. GNU Parallel zaistí, že sa jednotlivé úlohy vykonajú zároveň, čím sa šetrí čas potrebný na vykonanie všetkých úloh.

Kombinácia polí a GNU Parallel

Poslednou možnosťou je použitie poľa úloh spolu s GNU Parallel. Táto možnosť sa odporúča v takých prípadoch, kedy počet úloh, ktoré má GNU Parallel, presiahol maximálny limit úloh na užívateľa.

2.4.4 Software na clustroch

Pre vykonávanie rôznych úloh je veľmi často dôležité mať k dispozícii špecializovaný software ako napríklad Matlab, prípadne je často využívaná možnosť prekladania zdrojových súborov rôznymi prekladačmi, či rôznymi verziami toho istého prekladača.

Na clusteroch Anselm a Salomon je ponúkaný software organizovaný do tzv. modulov. Jedná sa o skupinu programov, ktoré sú si podobné, resp. sa využívajú pri vykonávaní rovnakých typov úloh. Ako príklad možno uviesť modul pozostávajúci z rôznych kompilátorov, ďalej modul debuggerov či prípadne modul s výpočtovými programami (Matlab, Octave a pod.). Pre automatizované vytváranie týchto modulov je použitý program *EasyBuild*.

EasyBuild umožňuje automatizovanú kompiláciu a inštaláciu softvéru. Spolieha sa na dve jeho súčasti, a to konkrétne *toolchains* a *EasyConfig súbor*.

Toolchain pozostáva z jedného alebo viacerých kompilátorov zvyčajne doplnených niekoľkými knižnicami pre špecifické funkcie. Jeho funkcia je skompilovať jednotlivé programy,

[7] viac informácii na oficiálnej stránke `http://www.gnu.org/software/parallel/`

ktoré sú súčasťou vytváraného modulu. Každý modul, ktorý sa vytvára musí mať uvedený použitý toolchain. Medzi podporované toolchainy patrí napríklad GCC, Intel, Clang, OpenMPI atď.

EasyConfig súbor slúži na podrobnú špecifikáciu pre EasyBuild. Skladá sa z textu Pythonovskej syntaxe, najčastejšie obsahuje príkazy priradenia, ktoré špecifikujú parametre modulu. Ukážka takéhoto súboru sa nachádza v prílohe C.1 [10]:

Kapitola 3

FabSim

FabSim je program, ktorý obsahuje sadu nástrojov pre automatizáciu práce so superpočítačmi. Jeho autorom je Derek Groen, prednášajúci na Brunel University London[1]. FabSim umožňuje vykonávať úlohy na superpočítačoch z prostredia príkazového riadku počítača užívateľa či prenos súborov medzi superpočítačom a počítačom, z ktorého je FabSim spúšťaný. Pre nastavenie premenných prostredia konkrétneho clustra je určený špeciálny súbor.

3.1 Dostupnosť a licencia

Fabsim je licencovaný pod trojbodovou BSD licenciou (BSD 3-clause license). Táto licencia vznikla zjednodušením štandardnej BSD licencie, preto sa veľmi často označuje prívlastkami „nová“ či „modifikovaná“[2].

Zjednodušenie sa týka odstránenia tretieho bodu z originálneho znenia BSD licencie, nazývaného aj „protivná reklamná BSD klauzula“. Táto klauzula prikazovala použitie špecifickej vety pri všetkých reklamných a propagačných materiáloch, v ktorých sa zmieňovalo použitie či funkcie daného softvéru. Vzhľadom k tomu, že táto klauzula bola často vývojármi menená a pôvodné slovné spojenie „University of California“ nahradzovali vlastnými menami, resp. názvami inštitúcií, dochádzalo k extrémnym prípadom, kedy program musel kvôli tejto klauzule obsahovať až 75 takýchto prehlásení [14]. Tieto udalosti nakoniec viedli k odstráneniu klauzly z licencie[3], čím vznikla spomínaná trojbodová verzia.

Na základe tejto licencie je možné program ľubovolne upravovať za predpokladu, že sa v zdrojových kódoch uvedie oznámenie o autorských právach, zoznam podmienok licencie a príslušné oznámenie. V prípade redistribúcie v binárnom formáte musia byť vyššie uvedené prehlásenia aj v dokumentácii k programu.
FabSim je verejne dostupný na stránke `http://www.github.com/djgroen/FabSim`.

3.2 Požiadavky na spustenie

Fabsim je implementovaný v programovacom jazyku Python 2.7.6. Python je interpretovaný jazyk vyššej úrovne. Znamená to teda, že jeho kód je ľahko čitateľný a zrozumiteľný pre človeka, je nezávislý na architektúre počítača na ktorom sa spúšťa a zdrojový súbor je pomocou interpreta priamo spustiteľný a vykonávaný po jednom riadku, bez nutnosti

[1] `http://people.brunel.ac.uk/~csstddg/`
[2] `https://opensource.org/licenses/BSD-3-Clause`
[3] `ftp://ftp.cs.berkeley.edu/pub/4bsd/README.Impt.License.Change`

kompilácie do strojového jazyka (jazyka nižšej úrovne)[3]. Z hľadiska ďalšieho členenia sa jedná o jazyk štrukturovaný, dynamický a objektovo orientovaný.

Syntax jazyka je veľmi jednoduchá, väčšina rezervovaných slov sú bežné anglické slová. Využíva sa odsadzovanie pomocou bielych znakov, čím sa zdrojový text delí do blokov a podblokov (jeden blok môže predstavovať napríklad definovaná trieda, v rámci nej môže byť podblok definovaná metóda či cyklus a pod.). Tento spôsob odsadzovania má za následok prehľadný a ľahko čitateľný kód. Z týchto a viacerých ďalších dôvodov je Python považovaný za jeden najvhodnejších programovacích jazykov pre začínajúcich programátorov[4].

Python má rozsiahlu štandardnú knižnicu, obsahuje však aj obrovské množstvo modulov, ktoré sú združované v Python Package Index (PyPI). PyPI je oficiálny repozitár, ktorý obsahuje moduly tretích strán. V súčasnosti sa v ňom nachádza viac ako 80 000 rôznych modulov[5]

V nasledujúcich podkapitolách si rozoberieme dva moduly, na ktorých je FabSim postavený.

3.2.1 PyYAML

PyYAML je modul, ktorý umožňuje vytváranie, modifikáciu a spracovanie YAML súborov.

YAML je formát slúžiaci na serializáciu dát, ktorý je zároveň ľahko čitateľný pre človeka. Z počiatku táto skratka znamenala anglické slovné spojenie „Yet Another Markup Language“, teda „ďalší značkovací jazyk“, avšak neskôr bol premenovaný na „YAML Ain't Markup Language“, v preklade „YAML nie je značkovací jazyk“.

Syntax jazyka YAML je inšpirovaná jazykmi C, HTML, Perl, či Python [16]. Jazyk taktiež podporuje spracovanie dátových typov používaných vo vyššie uvedených programovacích jazykoch, ako sú zoznamy alebo asociatívne polia.

Vo FabSime sa vyskytujú YAML súbory, v ktorých sú uložené dáta potrebné pre nastavenie prostredia superpočítačov. Medzi tieto dáta napríklad patria cesty k priečinkom s programami na spustenie či software, prípadne zoznam modulov, ktoré sa majú načitať po pripojení na cluster a zoznam bash aliasov. Tieto súbory taktiež obsahujú aj cesty na lokálnom počítači, napríklad cestu k priečinkom so zálohami, domovský adresár a pod.

3.2.2 Fabric

Najdôležitejším modulom, na ktorom je FabSim postavený, je jednoznačne Fabric. Už so samotného názvu vypovedá, akú kľúčovú rolu zohráva. Fabric je modul, ktorý umožňuje zjednodušené používanie SSH pre použitie v aplikáciách. Zjednodušene povedané, pomocou Fabricu je možné na diaľku obsluhovať príkazový riadok vybraného zariadenia pomocou SSH protokolu a vykonávať na ňom vlastné definované akcie, či už v normálnom móde, alebo v móde `sudo`. Tieto akcie možno vykonávať na viacerých zariadeniach súčasne.

Princíp funkcionality

Základom použitia je vytorenie Python súboru, v ktorom importujeme modul Fabric. Takýto súbor sa nazýva `Fabfile`. Vo Fabfile si nadefinujeme metódy podľa vlastnej potreby. Tieto metódy sa následne spúšťajú pomocou príkazu `fab nazov_metody`. Výhodou modulu Fabric je, že Fabfile môže obsahovať ľubovoľný kód Pythonu. Znamená to teda, že Fabfile

[4] https://docs.python.org/2/faq/general.html#is-python-a-good-language-for-beginning-programmers

[5] Uvedený počet platí k dátumu 17.5.2016, aktuálny stav je možné zistiť na adrese https://pypi.python.org/pypi

môže byť ľubovolný súbor, ktorý má v sebe importovaný modul Fabric. Dochádza tak k výbornej integrácii funkcií Fabricu a štandardného kódu Pythonu. Typickým príkladom tejto integrácie je prípad, kedy si v programe vytvoríme napríklad textový súbor, pomocou Fabricu ho prenesieme na druhé zariadenie a následne vzdialene spustíme program, ktorý tento súbor bude spracúvať.

Príkazy príkazového riadku, ktoré chceme vykonať na vzdialených počítačoch, vykonávame pomocou metódy `run`. Fabric poskytuje aj možnosť ovládania príkazového riadku lokálneho zariadenia a to konkrétne metódou `local`. Je síce pravda, že Python poskytuje náhrady príkazov v termináli (takéto metódy sú implementované napríklad v moduloch `os` alebo `sys`), svojími možnosťami sa však nedokážu rovnať rozsahu operácii, ktoré ponúka Fabric.

Environment - riadiace prostredie vo Fabric

Užívateľ, ktorý o Fabricu doteraz ešte nič nepočul, už po prečítaní predchádzajúcich riadkov má ucelenú predstavu, na akom princípe Fabric funguje a čo všetko sa s jeho pomocou dá spraviť. Núka sa však jedna zásadná otázka. Ako špecifikovať stroje, ktoré chceme vzdialene obsluhovať? Odpoveďou je Fabric environment.

Ako z názvu vyplýva, environment špecifikuje prostredie Fabricu a určuje jeho chovanie v rôznych situáciách. Z programátorského hľadiska sa jedná o štruktúru dátového typu asociatívne pole (v Pythone sa pre asociatívne pole používa aj termín `dictionary`, teda slovník). Vzhľadom k tomu, že riadi činnosť celého modulu, je implementovaný ako `Singleton`.

Environment teda obsahuje dvojice kľúč - hodnota, typické pre asociatívne polia. Kľúče nachádzajúce sa v tejto štruktúre sa nazývajú premenné prostredia. Fabric obsahuje veľké množstvo preddefinovaných premenných, avšak je možné si doň vložiť aj vlastné premenné (v prípade FabSimu sú to dáta z YML súborov, viď sekcia 3.2.1). Hodnoty preddefinovaných premenných je možné kedykoľvek meniť, odporúčaným postupom je však nemeniť tieto premenné nastálo, ale len pre tú časť kódu, v ktorej je to treba. Na túto činnosť sa najčastejšie používa kontextový manažér `settings`.

Vysvetľovať funkcionalitu každej premennej prostredia či metódy samotného modulu je pre čitateľa zbytočné a časovo náročné, pre záujemcov je však na mieste odporúčiť dokumentáciu k API Fabricu[6], ktorá je mimoriadne kvalitne a prehľadne spracovaná a ponúka komplexný prehľad vlastností a špecifík modulu Fabric.

3.3 Funkcie FabSimu

Pôvodná verzia FabSimu obsahuje systém pre nastavenie konfigurácie konkrétnych vzdialených zariadení (superpočítačov), interný jednoduchý spúšťač úloh či metódy na prenos súborov medzi lokálnym počítačom a vzdialeným zariadením. Medzi derivované verzie FabSimu patria moduly:

- FabHemeLB
- FabMD
- FabBioMD

[6] http://docs.fabfile.org/en/1.11/index.html

Tieto moduly využívajú vyššie uvedené funkcie FabSimu pre spúšťanie a riadenia simulácii,pričom sú prispôsobené na využívanie na superpočítačoch ARCHER, SuperMUC či BlueJoule. Moduly FabMD a FabBioMd sú súčasťou zdrojových kódov FabSimu, modul FabHemmeLB je vyvíjaný samostatne[7].

[7]Zdrojové kódy sú dostupné na adrese https://github.com/UCL/hemelb

Kapitola 4

Analýza a návrh softvéru z pohľadu softvérového inžinerstva

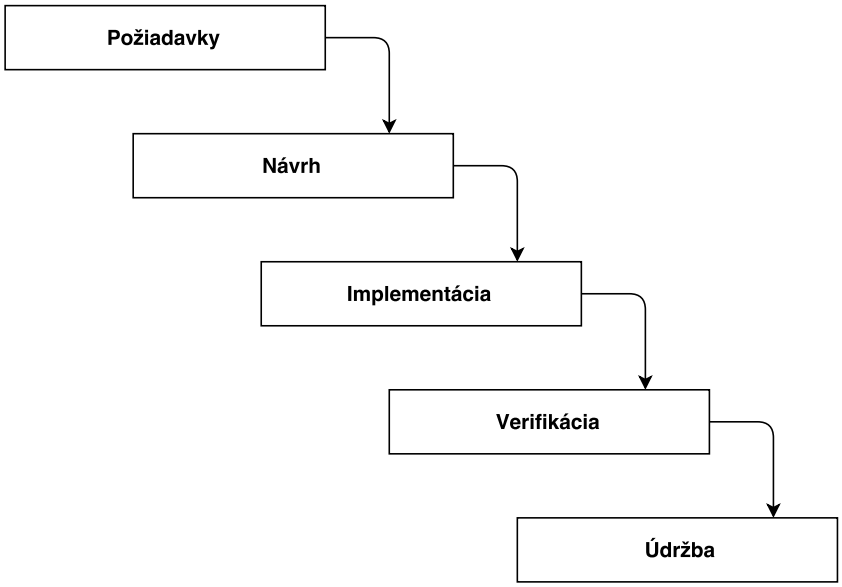
Ak chceme, aby výsledok našej práce bol dobrý, teda aby s ním bol zákazník spkojný a zároveň bola práca efektívna, či už časovo alebo vzhľadom k iným zdrojom, nemôžme sa hneď pustiť do programovania. V skutočnosti, samotné programovanie softvéru zaberá len veľmi malú časť (cca 12%) celého procesu. Tento proces sa nazýva *životný cyklus softvéru*. Práve *softvérové inžinierstvo* je odvetvie, ktoré sa zaoberá vývojom softvéru za použitia definovaných princípov, metód a postupov. Výsledkom softvérového inžinierstva je účinný a spoľahlivý softvérový produkt [15]. Pojem softérové inžinierstvo vznikol v 60. rokoch minulého storočia počas *softvérovej krízy*. Ako jej riešenie boli navrhnuté a predstavené tzv. *dobré programovacie praktiky* (best-practices), tzn. postupy, ktorých účinnosť je overená v praxi [9]. V nasledujúcej kapitole budú priblížené jednotlivé metódy a postupy a ich implementácia pri vývoji programu určeného k tejto práci.

4.1 Výber modelu životného cyklu softvéru

Hoci softvérové inžinierstvo definuje jednotlivé etapy a poradie v akom sa majú vykonať, existuje viacero prístupov z hľadiska výberu konkrétnej stratégie prístupu. Popíšeme si teda niektoré používané modely (paradigmy) a ich výhody a nevýhody voči ostatným modelom. Pre uvedenie do problematiky však nemá význam podrobne vysvetľovať každý typ, pretože ich v súčasnej dobe existuje príliš veľa a mnohé svojou náročnosťou na zdroje prevyšujú medze vyvýjaného softvéru pre túto prácu (nehovoriac o tom, že svojou komplikovanosťou môžu pri vývoji tak malého softvéru napáchať viac škody ako úžitku).

Vodopádový model

Vodopádový model je najzákladnejší a najjednoduchší model životného cyklu softvéru. Jednotlivé etapy nasledujú sekvenčne za sebou, tzn. ďalšia fáza začína až vtedy, keď predchádzajúca skončí. Nedochádza k návratu k predchádzajúcej etape. Práve táto vlastnosť spôsobuje problémy v momente, keď sa zistí že v niektorej z predchádzajúcich etáp došlo chybe, prípadne ak dôjde k zmenám požiadavok od zákazníka (k čomu v praxi dochádza veľmi často). V bežnej praxi sa preto používa len zriedka.



Obr. 4.1: Vodopádový model

Iteratívny model

Iteratívny model vychádza z vodpádového modelu. Z názvu vyplýva, že proces vývoja softvéru je rozdelený do iterácií, teda cyklického opakovania vodopádového modelu. Každá iterácia je samozrejme špecifická, jedná sa najmä o rozsah implementovanej časti softvéru. V prvej iterácii sa teda implementuje iba istá časť softvéru, ktorá sa predloží zákazníkovi. Po schválení a otestovaní sa začne pracovať na ďalšej iterácii, ktorá už obsahuje viac funkcií a takto sa pokračuje až pokiaľ nie je softvér úspešne implementovaný.

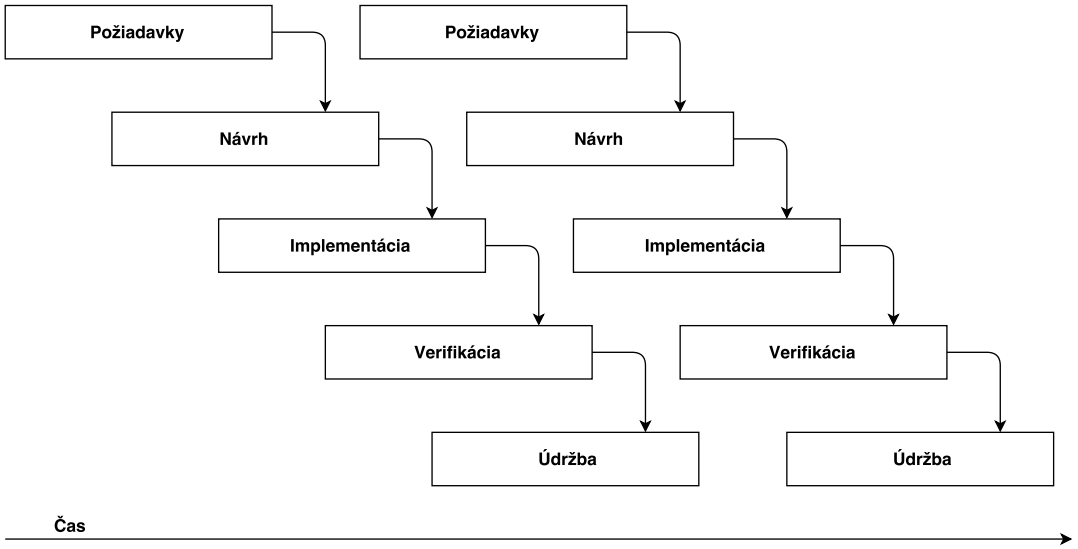
Výhodou oproti vodopádovému modelu je o poznanie väčšia fexibilita voči nečakaným zmenám či úpravám návrhu. Keďže dochádza k *dekompozícií* vývoja, teda deleniu na menšie časti, je postup vývoja prehľadnejší a ľahší na organizáciu. Nevýhodou je vyššia náročnosť na zdroje, či už časové alebo finančné.

Ďalšie používané modely

Medzi ďalšie populárne modely patrí napríklad **špirálový model**, ktorý vychádza z opakovaného vyvýjania prototypov, ktoré dostáva zákazník k otestovaniu. Po otestovaní sa tieto prototypy zahodia a pokračuje sa na vývoji ďalšieho prototypu s rozšírenou funkcionalitou.

Alternatívu k vodopádovému modelu predstavuje **V-model**, pri ktorom v každej fáze paralelne navrhne a pripraví sada testov. Tieto testy neskôr nasledujú v opačnom poradí ako boli vytvorené, čo dodáva diagramu tvar písmena V. Nazýva sa aj *model verifikácie a validácie*.

Špeciálnym typom modelu (pojem model však treba brať značne s nadhľadom) je **Big Bang model**. Tento model sa opiera o čo najviac programovania a čo najmenej plánovania a analýz. Nie je stanovený žiaden presný postup vývoja, preto sa neodporúča tento model využívať a je skôr len extrémnym príkladom prístupu k vývoju softvéru.



Obr. 4.2: Iteratívny model

4.2 Analýza zadania a získavanie informácií

Úlohou tejto práce je vytvoriť modul do už existujúceho programu FabSim, ktorý by umožňoval plnohodnotnú obsluhu clusterov Anselm a Salomon. Tento návrh pochádza od vedúceho práce, ktorý je sám členom výskumného tímu, ktorý na daných clusteroch pracuje. Clustre sa používajú pri spúšťaní časovo i hardvérovo náročných simulácií určených na spätnú rekonštrukciu pulzných vĺn vystrelených do tkaniva človeka.

Tieto simulácie sa vykonávajú spúšťaním už hotových programov s rôznymi vstupnými súbormi. K dispozícii je už teda hardvér, rovnako aj softvér, ktorý sa na ňom bude spúšťať. Cieľom tejto práce je preto zaobstarať automatizované vytváranie a spúšťanie týchto simulácii na základe vstupu od užívateľa. Dôležitou vlastnosťou je aj jednoduché riadenie a obsluha prostredia superpočítača, intuitívny prenos súborov medzi lokálnym zariadením a clusterom a možnosti pokročilej správy súborov pomocou systému riadenia revízií Git.

Hoci FabSim už obsahuje implementované metódy pre spúšťanie simulácií, jedná sa o príliš špecifickú implementáciu zameranú na také typy simulácií, ktoré zadávatel úlohy nevyužije.

Z hľadiska dizajnu sa má jednať o konzolovú aplikáciu. Táto možnosť bola zvolená z toho dôvodu, že sa v budúcnosti plánuje vytvorenie aplikácie s grafickým užívateľským rozhraním, ktoré by plne automatizovane spúšťalo simulácie pomocou FabSimu na základe výsledkov a analýzy snímkov magnetickej rezonancie mozgu človeka. V tejto dobe preto nie je nutné zaoberať sa dizajnom užívateľského prostredia.

4.3 Koncoví užívatelia produktu

Počet užívateľov produktu sa dá rátať v jednotkách, maximálne desiatkach ľudí. Jedná sa o úzko špecifikovaný softvér šitý na mieru. Užívatelia patria do skupiny ľudí, ktorí nemajú vzdelanie v obore informatiky a prácu s počítačom ovládajú len na základnej, prípadne mierne pokročilej úrovni. V kombinácií s požiadavkou na konzolovú aplikáciu sa jedná o náročnú úlohu, pri ktorej je potreba zladiť dohromady všetky požiadavky a zaistiť aby

príkazy, ktoré softvér riadia, boli dostatočne jednoduché a zrozumiteľné aj pre človeka bez vzdelania v tomto obore.

Keďže drvivá väčšina užívateľov nežije v Českej republike, nebolo možné zaistiť adekvátne osobné stretnutie za účelom bližšej špecifikácie požiadavok. Požiadavky sa preto tlmočili cez vedúceho práce, ktorý je jediným členom tohto výskumného tímu so vzdelaním v oblasti informatiky. Dokáže teda viac než dobre posúdiť, ktoré požiadavky sú prínosné a ktoré, naopak, nie je nutné implementovať. Z tohto titulu preto absencia stretnutia so zvyšnými užívateľmi nespôsobila žiadnu ujmu na kvalite spracovania požiadavok.

4.4 Management zdrojov

Vzhľadom k tomu, že táto práca je vykonávaná individuálne, nie je nutné používať všetky vybrané princípy a postupy projektového managementu. Za zmienku stojí len *organizácia zdrojov projektu*. Za zdroj sa považuje každý prvok, ktorý môže byť použitý pri vývoji softvéru. Môže sa jednať napríklad o rozpočet, ľudské zdroje či nástroje určené na implementáciu. Tieto zdroje sú dostupné v obmedzenom množstve a ich nedostatok bráni vývoju projektu [15].

Ak sa zamyslíme nad účelom a rozsahom tejto práce, zistíme, že jediným zdrojom, ktorým disponujeme, je čas. O to efektívnejšie je však potreba ho využiť a naplánovať jednotlivé etapy zo životného cyklu softvéru (viď kapitola 4.1) tak, aby sa zaistil hladký priebeh implementácie.

Na grafickú vizualizáciu naplánovania postupnosti jednotlivých činností v čase sa využíva *Ganntov diagram*. V projektovom managemente sa používa jeho upravená verzia, ktorá zohľadňuje aj závislosti medzi jednotlivými entitami grafu [1].

Pri zostavovaní grafu sa vychádzalo z nasledujúcich údajov:

- Dátum začiatku práce je 8.2.2016. Tento dátum bol zvolený preto, lebo práve od začiatku letného semestra na FIT VUT mal autor dostatok času na venovanie všetkých síl a koncentrácie na túto prácu.
- Dátum konca práce je 16.5.2016. Hoci reálny dátum odovzdania práce je určený neskôr, z dôvodu časovej rezervy na ďalšie úkony potrebné na odovzdanie práce a softvéru boli na tieto účely vyhradené dva dni.
- Na vykonanie práce bolo teda vyhradených 14 týždňov. Jeden týždeň predstavuje 5 pracovných dní, z toho v každom boli na tento projekt vyhradené 4 hodiny. Dohromady to teda dáva 280 naplánovaných hodín.

Výsledný časový harmonogram je možné vidieť v grafe B.1.

Kapitola 5

Postup implementácie

Po predchádzajúcej kapitole máme prehľad o tom, čo chceme vytvoriť, máme zozbierané požiadavky od zákazníka, určenú taktiku prístupu (model životného cyklu softvéru) a takisto časový harmonogram prác. Len pre úplnosť dodajme, že zvolený bol iteratívny model. Jednotlivé iterácie predstavovali vždy čas medzi dvoma konzultáciami s vedúcim práce, kedy sa prezentovali výsledky za predchádzajúce obdobie a zároveň sa plánoval rozsah prác pre ďalšiu iteráciu. Postup implementácie tak možno rozdeliť na niekoľko častí, ktoré si v nasledujúcich sekciách priblížime.

5.1 Zoznámenie sa s prostredím a prvé metódy

Ešte pred samotným programovaním bolo nutné podrobne sa zoznámiť so súčasnou verziou FabSimu a so špecifikami modulu Fabric. Keďže súčasná verzia FabSimu neobsahuje žiadnu dokumentáciu, len zoznam príkazov s veľmi stručným popisom (zvyčajne sa jedná o jednu jednoduchú vetu), prebiehali prvé spúšťania programu metódou pokus-omyl.

Po tomto náročnom začiatku boli ako prvý cieľ stanovené metódy pre prenos súborov, ktoré sa neskôr využívali pri ostatných metódach. Naštudovaním dokumentácie k Fabricu bolo zistené,že tieto metódy sú priamo podporované a došlo teda len k ich veľmi malej úprave.

5.2 Spúšťanie simulácií

Ďalším krokom bolo zautomatizovanie spúšťania simulácii na clustri. Pôvodná myšlienka bola spúšťať vzdialene len jeden príkaz, ktorý na základe užívateľského vstupu zadá príkaz pre zaradenie úlohy do fronty. Po konzultácii s vedúcim práce sa však nakoniec presedlalo na alternatívnu metódu, ktorou bolo vytvorenie PBS skriptu a následne jeho prenos na vzdialné zariadenie a spustenie.

PBS skripty majú zo značnej časti podobnú štruktúru. Najvhodnejším prístupom tak bolo vytvorenie predlohy (templatu) PBS skriptu, ktorá by bola dostupná v zdrojových súboroch a ktorej obsah by sa upravoval podľa špecifík konkrétnej úlohy, ktorú chce užívateľ realizovať. Tento prístup bol realizovaný pomocou modulu `jinja2`[1]. Takýto prístup umožňuje dynamickú tvorbu PBS skriptov, kedy sú na základe užívateľského vstupu zmenené príslušné hodnoty PBS skriptu (napríklad počet alokovaných jadier a pod.). Predloha obecného PBS skriptu je zobrazená v ukážke kódu 5.1.

[1] Viac informácií na `http://jinja.pocoo.org/docs/dev/`

```
"#!/bin/bash
#PBS -q {{ queue }}
#PBS -A {{ project }}
#PBS -l select={{ nodes }}:ncpus={{ cpus }}:mpiprocs={{ mpi }}:
    ompthreads={{ omp }}
#PBS -l walltime={{ walltime }}
#PBS -m be
#PBS -N KW-L25

export OMP_PROC_BIND=true
export OMP_PLACES=cores

#load modules

cd $PBS_O_WORKDIR
./{{ program }} > {{ output }}
```

Listing 5.1: Predloha obecného PBS skriptu

Po úspešnom implementovaní vytvárania PBS skriptov a ich spúšťania bolo už veľmi jednoduché vytvoriť podobné metódy aj pre úlohy s využívajúce Matlab či úlohy typu job array a GNU Parallel. Tieto metódy sú si veľmi podobné, rozdiel však predstauje o niečo iný obsah PBS skriptu (z toho dôvodu má každý typ úlohy svoju vlastnú predlohu PBS skriptu) a z toho vyplývajúce rozdielne požiadavky na užívateľský vstup. Príklad takéhoto PBS skriptu je možné vidieť na ukážke kódu 5.2.

```
#!/bin/bash
#PBS -q qexp
#PBS -A OPEN-5-13
#PBS -l select=1:ncpus=24:mpiprocs=0:ompthreads=0
#PBS -l walltime=00:01:00
#PBS -m be
#PBS -N KW-L25

export OMP_PROC_BIND=true
export OMP_PLACES=cores

module load matlab/R2013a-EDU

cd $PBS_O_WORKDIR
matlab -nodisplay -r matlab > output
```

Listing 5.2: PBS skript pre prácu s Matlabovskými úlohami

5.3 Rozšírenia

Po zautomatizovaní vykonávania simulácií bolo účelom zaistiť čo najjednoduchšie používanie softvéru a vytvoriť sadu metód, ktoré síce nie sú kritické z hľadiska funkčnosti softvéru,

avšak poskytujú dostatočný užívateľský komfort. Patria sem napríklad metódy pre integráciu práce s Gitom (7.3) či riadiace a organizačné metódy (7.4).

Implementácia metód pre prácu s Gitom bola pomerne jednoduchá, kedy stačilo len vzdialene spúšťať príslušné prikazy Gitu a dômyselne vymyslieť spôsob vstupných parametrov. Pri riadiacich a organizačných metódach sa už vyžadovalo viac kreativity, kedy bolo nutné sa vžiť do role užívateľa a prispôsobovať ich jeho potrebám. Boli tak vytvorené metódy, ktoré sú schopné zálohovať či obnoviť dáta na clustri a to pomocou skôr implementovaných metód pre prenos súborov.
Osobitnou kapitolou je metóda pre skontrolovanie konfiguračných súborov. Keďže človek je tvor omylný, často dochádza k zlyhaniu ľudského faktora, zvlášť u ľudí, ktorí nemajú dostatočné vzdelanie v danom obore. V rámci zamedzenia takýchto chýb z nepozornosti sa kontrolujú jednotlivé položky konfiguračných súborov a užívateľ je upozornený o nevyplnených položkách.

Ako posledné boli implementované pomocné metódy, napríklad metóda na nainštalovanie knižnice hdf5.

5.4 Dokumentácia

Veľmi dôležitou časťou celého projektu je aj podrobná dokumentácia k softvéru. Práve tá bola najväčšou slabinou FabSimu, keďže obsahovala len zoznam príkazov bez akéhokoľvek vysvetlenia ich použitia. Na druhú stranu však treba dodať, že FabSim bol používaný len veľmi málo ľuďmi a teda nebol dôvod písať k nemu dokumentáciu, pretože ľudia, ktorí ho používali si vedeli význam jednotlivých metód vyvodiť zo zdrojových kódov.

Ako už bolo spomínané, úloha je však koncipovaná pre ľudí, ktorí nemajú vzdelanie v obore informatiky a preto táto možnosť „dokumentácie“ je pre nich absolútne neprijateľná. V kapitole 7 je preto spísaný zoznam všetkých implementovaných metód rozdelených podľa kategórii a ku každej z nich je napísaný stručný popis a príklady spustenia.

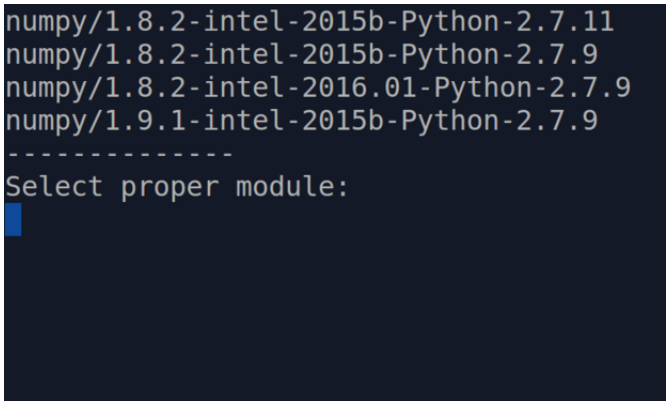
5.5 Popis chovania programu

Doteraz popísané postupy môžu pre nejedného čitateľa znieť až príliš abstraktne. V nasledujúcej sekcii bude preto podrobne rozpísané chovanie programu po spustení príkazu `autojob`. Táto metóda bola zvolená preto, lebo obsahuje najširšie spektrum príkazov a teda poskytuje najväčší priestor pre ukážku toho, čo sa počas behu programu udeje. Presný popis metód, ktoré spúšťa, ako aj príklady spustenia je možné nájsť v kapitole 7.2. Kód, ktorý sa bude spúšťať na clustri je jednoduchý program napísaný v jazyku C++.

Po zadaní príkazu `fab salomon autojob:example/example.cpp` sa teda vykonajú nasledujúce príkazy:

1. Spustí sa metóda `prepare_job`. V nej sa upraví predloha PBS skriptu z ukážky kódu 5.1 na finálnu podobu, ktorú možno vidieť v prílohe C.3. Užívatelovi je ponúknutá možnosť spustenia modulov podľa jeho uváženia. Vyhľadávajú sa všetky moduly, ktoré obsahujú zadanú frázu (viď obrázok 5.1, v ktorom bol vyhľadávaný modul `numpy`).

2. Celý obsah zložky `example` sa premiestni na cluster. Stane sa tak vykonaním príkazu `put('%s/%s/*'%(env.local_programms_path, folder), '%s/%s' % (env.remote_programms_path, folder))`. Premenné `env.local_programms_path` a `env.remote`

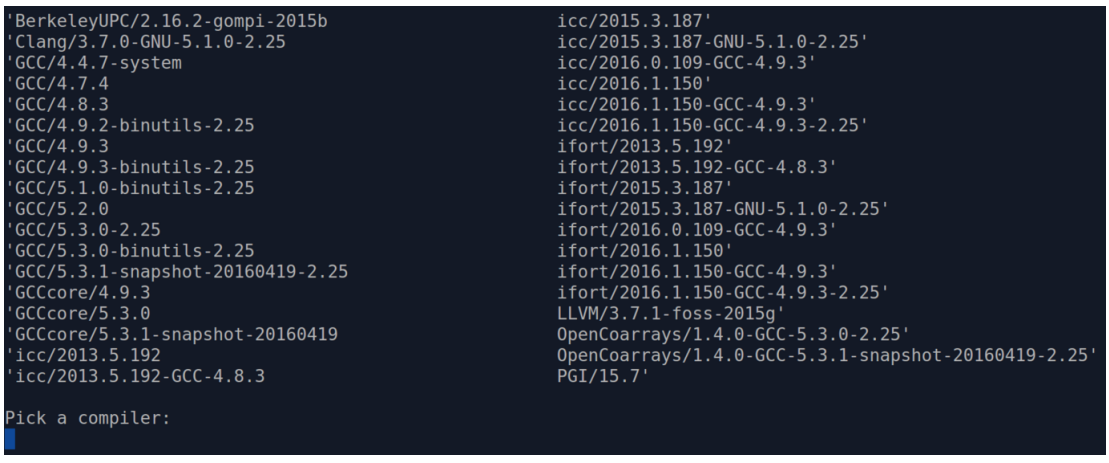


Obr. 5.1: Výber príslušného modulu na základe vyhľadávania

`_programms_path` sú vysvetlené v kapitole 7.1. Pod premennou `folder` sa rozumie adresár, v ktorom je spúšťaný program umiestnený, v našom prípade sa jedná o adresár `example`.

3. Keďže sa jedná o program jazyka C++, musí prebehnúť kompilácia. Obrázok 5.2 zobrazuje ponuku všetkých dostupných kompilátorov na clustri. Je na užívateľovi, ktorý z nich si vyberie. Po výbere kompilátora sa vykoná séria troch príkazov:

   - Ako prvý sa vykoná príkaz `cd env.remote_programms_path/folder`. Deje sa tak preto, lebo pri každom vzdialenom pripojení na cluster sa FabSim ocitne v domovskom priečinku. Nemôže teda dôjsť k okamžitej kompilácií súboru, lebo ten sa nachádza v inom adresári.
   - Načíta sa zvolený kompilátor pomocou príkazu `module load`
   - Prebehne kompilácia súboru. Každý kompilátor používa svoje špecifické príkazy. Konkrétne pri použitom kompilátore `GCC/4.7.4` to bude príkaz `g++ -Wall -Wextra -pedantic -O2 example.cpp -o example`.



Obr. 5.2: Ponuka kompilátorov nachádzajúcich sa na

4. Vykoná sa metóda `submit_job`. Keďže na jej vstupe bude vygenerovaný PBS skript, vykonajú sa príkazy `cd env.remote_programms_path/folder` a `qsub ./jobscipt _name`.

V tomto okamihu bol spustený PBS skript a program ktorý spúšťa sa zaradil do fronty. Užívateľ takisto obdržal e–mail o tom, že program bol zaradený do fronty. Po skončení behu programu obdrží ďalší informačný mail. Tieto maily obsahujú názvy súborov vygenerovaných plánovačom a ďalšie deatily o priebehu. Takéto súbory sa generujú vždy dva, jeden obsahuje výstupy programu a druhý chybové hlášky. Chovanie programu vyjadrené v diagrame je možné pozorovať na obrázku B.2.1

Kapitola 6

Testovanie

Testovanie je dôležitou časťou vývoja softvéru. Obecne je testovanie definované ako činnosť, pri ktorej je systém alebo komponent vykonávaný za vopred stanovených podmienok, pričom výsledky sú pozorované a zaznamenávané a vyhodnotenie je založené na základe určitého aspektu systému alebo komponentu [13]. V prípade testovania softvéru sa jedná o proces, resp. sadu procesov, ktoré sú navrhnuté tak, aby sme si boli istí že program robí to, na čo bol navrhnutý a naopak, že nerobí nič čo nechceme [11].

Aj keby sme uvažovali, že pôvodná verzia FabSimu neobsahuje žiadne chyby a funguje bez problémov, pri každej úprave kódu programu sa zvyšuje šanca, že sa v ňom vyskytnú nejaké nové chyby. Vzhľadom k tomu, že do FabSimu je integrovaný úplne nový modul, riadny proces testovania má skutočne svoje opodstatnenie. Keďže FabSim je používaný pomerne úzkou skupinou ľudí (viď kapitola 4.3), je dôležité získať od nich istú spätnú väzbu, aby bol zaistený čo najväčší komfort pri používaní.

Ešte predtým, ako si priblížime zvolené postupy a výsledky testovania, zameriame sa na rôzne spôsoby testovania a ich výhody a nevýhody.

6.1 Metódy testovania

Na testovanie sa dá prihliadať z viacerých uhlov pohľadu. Vo všeobecnosti rozlišujeme *white-box* a *black-box* testovanie. Pri white-box testovaní má tester znalosti o princípe činnosti programu a navyše má k dispozícii aj zdrojový kód softvéru. Naopak pri black-box testovaní, tester nevidí dovnútra softvéru, teda nemá k zdrojovým kódom prístup.

Pri white-box testovaní sa používajú ďalšie dve zaužívané metódy. Jedna sa o *statickú analýzu* a *dynamickú analýzu* kódu. Pri statickej analýze dochádza k testovaniu formou prezerania kódu a snahy nájsť sémantické či logické chyby. Formou statickej analýzy je aj *inšpekcia kódu*, kedy sa kód porovnáva so zaužívanými metódami (tzv. *best practices*). Dynamická analýza kódu sa odohráva počas behu programu a jej cieľom je zistiť napríklad či nedochádza k nadmernému zaťažovaniu procesora či pamäte [6].

Z hľadiska ľúdí, ktorí softvér testujú, môžeme hovoriť o *alfa* a *beta* testovaní. Alfa testovanie vykonávajú vývojári, prípadne nezávislý tím testerov (stále sa však jedná o interných testerov, teda v rámci vývojárskej firmy). Tým pádom spadá pod white-box a aj pod black-box testovanie. Beta testovanie (nazýva sa aj *Field testing*) naopak vykonávajú koncoví užívatelia a patrí iba pod black-box testovanie.

Častou používanou metódou (napriek tomu že sa v mnohých prípadoch jedná o metódu náročnú na čas aj zdroje) je aj *regresné testovanie*. Regresné testovanie pracuje na princípe

opakovania testov po každej úprave kódu programu aby sa zistilo, či táto zmena nespôsobila chyby v takej časti kódu, ktorá nebola zmenená [17].

6.2 Priebeh testovania

Keďže program, resp. modul implementovaný v rámci tejto práce, bol vyvýjaný jedným človekom, nebolo možné technicky ani časovo vykonať testovanie takým spôsobom, ako bolo popísané v predchádzajúcej sekcii. Napriek tomu však bolo cieľom riadne otestovať program pomocou viacerých metód a pokiaľ možno čo najviac dodržať zaužívané postupy.

Testovanie prebiehalo počas celej doby implementácie. Po implmenetovaní každej metódy došlo k otestovaniu na clustri Salomon aby sa zistilo, či sa program chová podľa očakávaní. Vzhľadom k tomu, že clustre Salomon a Anselm sú si veľmi podobné, v prvej fáze prebiehalo testovanie len na jednom clustri. Po dokončení jedného bloku metód (napr. všetkých metód potrebných k vytvoreniu a spusteniu klasických úloh) bola znova otestovaná funkčnosť každej jednej z nich. Statická analýza kódu bola implementovaná pomocou programu Pyflakes[1].

Po tejto fáze, v čase, keď už bola implementovaná celá funkcionalita modulu, došlo k druhému kolu testovania, v ktorom bola snaha vžiť sa do role užívateľa a odhaliť čo najviac chýb, ktoré by mohli vzniknúť nedopatrením alebo iným zlyhaním ľudského faktora. Došlo teda k ošetreniu chybových stavov (napríklad pri zvolení neexistujúceho súboru) a k lepšiemu formátovaniu chybových hlášok. Zamedzilo sa tak neočakávaným chovaniam programu a zároveň sa zlepšilo užívateľské prostredie, keďže prípadné chyby sú jednoznačne interpretované a zároveň je v každej chybovej hláške napísaná pravdepodobná príčina chyby.

V tomto okamihu bola zabezpečená funkčnosť na clustri Salomon a boli ošetrené chybové stavy. Nasledovalo testovanie na clustri Anselm. Touto fázou bolo ukončené alfa testovanie (a samozrejme aj white-box testovanie).

Beta testovanie spočívalo z dvoch častí. Prvou bolo predanie programu vedúcemu práce (ktorý bude tiež užívateľom), ktorý program otestoval a následneho ho poskytol ďalším koncovým užívateľom pre poslednú fázu testovania.

[1] Viac informácií na https://pypi.python.org/pypi/pyflakes

Kapitola 7

Manuál k upravenej verzii Fabsimu

Implementované metódy v novom module FabSimu možno rozdeliť do troch kategórií a to

- Metódy pre vytváranie a spúšťanie úloh
- Metódy pre prácu s Gitom
- Riadiace a organizačné metódy

V nasledujúcich sekciách si priblížime spôsoby použitia týchto metód, rovnako ako princípy implementácie. Ešte predtým je však treba vysvetliť princíp a použitie konfiguračných súborov.

7.1 Konfiguračné súbory

FabSim obsahuje dva konfiguračné súbory, a to konkrétne súbory `machines.yml` a `machines_user.yml`. Princíp ich činnosti bol vysvetlený v sekcii 3.2.1. Z licenčných dôvodov nie je možné meniť obsah súboru `machines.yml`. Po miernej úprave kódu však bolo implementované riešenie, kedy je možné zo súboru `machines_user.yml` získavať vyplnené údaje rovnako ako v zo súboru `machines.yml` (súbor `machines_user.yml` slúžil pôvodne len na prepisovanie hodnôt v `machines.yml`, z dôvodu rozdielnych zariadení to však už nie je možné). Teraz si predstavíme najdôležitejšie položky týchto súborov a popíšeme ich význam:

- **username** - prihlasovacie meno k jednotlivému zariadeniu
- **project** - názov projektu, resp. pracovnej skupiny, ku ktorej užívateľ patrí. Táto položka je dôležitá pri vytváraní úloh, aby bolo jasné, ku ktorému tímu užívateľ patrí aby sa vedel prirátať alokovaný procesorový čas.
- **remote** - adresa zariadenia
- **key_filename** - cesta k súboru obsahujúci ssh kľúč. Tento súbor je potrebný na identifikáciu pri prihlasovaní sa k väčšine superpočítačov
- **remote_home_path** - cesta k domovskému adresáru na vzdialenom zariadní. Tento priečinok by sa mal vyskytovať na diskovom poli `SCRATCH` v adresári `WORK`[1].

[1] https://docs.it4i.cz/salomon/storage#section-12

- **remote__programms__path** - cesta k adresáru, ktorý obsahuje zdrojové kódy programov určených k simulácii. Podobne ako `remote\_home\_path` by sa mal nachádzať na diskovom poli `scratch`.
- **local__programms__path** - adresár na počítači užívateľa, ktorý obsahuje programy, ktoré sa neskôr budú spúšťať na superpočítačoch.
- **backup__path** - cesta k adresáru, do ktorého sa budú ukladať zálohy súborov zo superpočítačov.
- **local__home__path** - domovská zložka FabSimu. Jedná sa vždy o cestu na konci ktorej je adresár `deploy`.
- **git** - adresa ku git repozitáru užívateľa. Táto položka nie je povinná, teda užívateľ ju nemusí vyplniť. Ak však chce využívať všetky výhody FabSimu, odporúča sa vyplniť aj toto pole
- **git__path** - adresa k adresáru, ktorý obsahuje stiahnutý repozitár. Táto cesta slúži k tomu, aby sa pri metódach pracujúcich s gitom (napr. `git_push`) nemusela zadávať celá cesta k súboru, ale začínalo sa až od tohto priečinka.
- **aliases** - zoznam užívateľom definovaných aliasov. Táto položka má tvar dátovej štruktúry slovník (resp. asociatívne pole) kde kľúčom je názov aliasu a hodnotou je príkaz ktorý sa vykoná po zadaní tohto aliasu. Obe hodnoty musia byť dátového typu `string`, teda musia byť ohraničené úvodzovkami. Príklad použitia: {“home”:“cd /scratch/xstrec05”}. Táto položka nie je povinná.
- **modules** - zoznam modulov, ktoré sa načítajú vždy po pripojení na cluster. Táto položka má tvar dátovej štruktúry zoznam a jednotlivé hodnoty musia byť dátového typu `string` a oddelené čiarkami. Táto položka nie je povinná.
- **job__details** - jedná sa o štruktúru asociatívneho poľa, ktorá obsahuje preddefinované hodnoty používané pri zaraďovaní úloh do fronty. Tieto hodnoty slúžia ako predvolené hodnoty volitelných parametrov metód, ktoré plánujú úlohy. Explicitným špecifikovaním jednotlivých parametrov pri volaní týchto metód sa ich hodnoty prepíšu a požijú sa tie zo vstupu. Táto štruktúra obsahuje nasledujúce prvky:
  - **queue** - názov fronty ktorá sa použije ako predvolená pri plánovaní úloh
  - **cpus** - predvolený počet CPU, používa sa pri plánovaní úloh
  - **nodes** - predvolený počet uzlov, používaný pri plánovaní úloh
  - **walltime** - predvolená alokovaná doba pri plánovaní úloh
  - **omp** - predvolený počet omp uzlov
  - **mpi** - predvolený počet mpi uzlov
  - **output** - predvolený názov výstupného súboru

Každé zariadenie by malo obsahovať tieto položky (minimálne všetky povinné). Syntax konfiguračných súborov je nasledovná: ako prvý sa uvádza názov vzdialeného zariadenia ukončený dvojbodkou. Pod ním nasledujú vyššie uvedené položky ktoré sú odsadené o dĺžku jedného tabulátora. Názvy týchto položiek sú od ich hodnôt oddelené dvojbodkou. Napriek tomu, že názvy položiek sú dátového typu `string`, netreba ich uvádzať do úvodzoviek. Ukážku správne vyplneného konfiguračného súboru je možné vidieť v prílohe C.2

7.2 Metódy pre vytváranie a spúšťanie úloh

Ako už z názvu vypovedá, do tejto skupiny implementovaných metód patria také, ktoré sa nejakým spôsobom podieľajú na vyvtáraní a spúšťaní úloh. Tieto úlohy môžu spúšťať všeobecne rôzne programy, implementované sú však aj metódy, ktoré vytvárajú a spúšťajú Matlabovské úlohy, polia úloh či skripty využívajúce technológiu GNU Parallel.

Ešte predtým, ako si priblížime jednotlivé metódy, vysvetlíme význam jednotlivých parametrov, ktoré sa pri týchto metódach používajú. Väčšina metód má parametre rovnaké alebo veľmi podobné, preto by bolo zbytočné ich pri každej jednej zvlášť vysvetľovať.

Zoznam používaných parametrov

- **program** - cesta k programu, ktorý sa bude spúšťať v simulácii (môže sa jednať aj o ešte neskompilovaný zdrojový súbor)
- **queue** - názov fronty, do ktorej sa zaradí príslušná úloha
- **nodes** - počet uzlov, ktoré sa alokujú pre exekúciu úlohy
- **cpus** - počet CPU, ktoré sa alokujú na každom uzle pre exekúciu úlohy
- **walltime** - vyhradený procesorový čas. Má tvar `hh:mm:ss`
- **omp** - počet omp jadier, ktoré sa alokujú pre potreby exekúcie úlohy
- **mpi** - počet mpi jadier, ktoré sa alokujú pre potreby exekúcie úlohy
- **file** - cesta k programu, ktorý sa má skompilovať
- **path** - adresár, v ktorom sa nachádza súbor ktorý sa zaradí do fronty (resp. adresár, v ktorom sa nachádza PBS skript)
- **pbs** - názov príslušného PBS skriptu (nepovinný parameter)
- **input_prefix** - začiatok názvu vstupných súborov
- **jobname** - názov úlohy, ktorá sa zaradí do fronty
- **begin** - číslo prvého vstupného súboru
- **end** - číslo posledného vstupného súboru
- **job_nr** - identifikačné číslo úlohy zaradenej vo fronte
- **output** - názov výstupného súboru, do ktorého sa zaznamenáva priebeh simulácií

Po oboznámení s používanými parametrami nasleduje zoznam implementovaných metód spolu s ich popisom a príkladmi použitia.

prepare_job

Alias: `pj`
Vstupné parametre: povinný parameter `program`, volitelné parametre `queue`, `nodes`,

`cpus`, `walltime`, `omp`, `mpi`, `output`.
Popis: Táto metóda vytvorí na základe vstupných parametrov PBS script pre všeobecnú úlohu. Povinný parameter `program` špecifikuje cestu k súboru od adresára uloženého v premennej prostredia `env.local_programms_path`. Ak je v parametri `program` zadaný zdrojový súbor (napríklad súbor s koncovkou .cpp), ako spustitelný program sa uvažuje binárny súbor s rovnakým názvom bez prípony[2].
Počas vytvárania PBS skriptu je užívateľovi ponúknutá možnosť zvolenia modulov, ktoré sa majú pred zaradením programu do fronty načítať (princíp vyhľadávania je vysvetlený v sekcii 7.5). Po vytvorení PBS skriptu sa celá zložka premiestni na vzdialené zariadenie. Ak príslušná adresárová cesta neexistuje, vytvorí sa[3].
Príklady použitia:
```
fab salomon prepare_job:programms/program.cpp
fab salomon prepare_job:programms/program.cpp,queue=qprod
```

compile_file

Alias: `cf`
Vstupné parametre: povinný parameter `file`.
Popis: Metóda, ktorá skompiluje príslušný súbor. Cesta k súboru v parametri `file` začína od adresára v riadiacej premennej `env.home_programms_path`. Počas tohto procesu je užívateľovi poskytnutý výpis aktuálnych kompilátorov na clustri a je vyzvaný na výber jedného z nich. Ak je kompilátor schopný zdrojový súbor preložiť, vytvorí sa jeho binárny súbor. V opačnom prípade sa vypíše príslušná chybová hláška a proces sa ukončí.
Príklady použitia:
```
fab salomon compile:programms/prog1.cpp
```

submit_job

Alias: `sj`
Vstupné parametre: povinný parameter `path`, voliteľné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`, `program`, `pbs`, `output`
Popis: Na základe zadaných parametrov sa vloží do príslušnej fronty nová úloha. Platí, že aspoň jeden z parametrov `program` alebo `pbs` musí byť vyplnený (predvolene majú oba hodnotu `None`). Takisto platí, že parameter `pbs` má väčšiu prioritu, ak je teda zadaný v kombinácii s inými voliteľnými parametrami, odignorujú sa. Cesta k súboru v parametri `file` začína od adresára v riadiacej premennej `env.remote\_programms\_path`.
Príklady použitia:
```
fab salomon submit_job:programms,pbs_script_name
fab anselm submit_job:programms,queue=qprod,program=prog1
```

autojob

Alias: `aj`
Vstupné parametre:povinný parameter `program`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`, `output`.
Popis: Táto metóda vykonáva plne automatizované vytvorenie a zaradenie úlohy do fronty.

[2] Napr. pri zadaní súboru `program.cpp` sa uvažuje že jeho spustitelný súbor je `programm`.
[3] Príslušné priečinky opäť začínajú v adresári `env.remote_programms_path`.

V podstate vykoná za sebou metódy `prepare_job`, `compile_file`[4] a `submit_job`.
Príklady použitia:
```
fab salomon autojob:programms/program.cpp
fab salomon aj:programms/program.cpp,queue=qprod
```

prepare_matlab

Alias: `pm`
Vstupné parametre:povinné parametre `program`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`, `output`.
Popis: Metóda pracuje na podobnom princípe ako `prepare_job`. Užívateľ je pred vytvorením PBS skriptu vyzvaný, aby si vybral zo zoznamu verziu Matlabu, s ktorou chce pracovať. Následne môže užívateľ zvoliť ďalšie moduly, ktoré úloha vyžaduje (viď sekcia 7.5). Po vytvorení PBS skriptu sa celá zložka, v ktorej sa zdrojový súbor nachádza, skopíruje na vzdialené zariadenie. Ak daná adresárová cesta neexistuje, automaticky sa vytvorí.
Pozn.: Manuálne spúšťanie PBS skriptu vytvoreného touto metódou je možné pomocou `submit_job`.
Príklady použitia:
```
fab salomon prepare_matlab:programms/Matlab/program.m,outputfile.txt
fab salomon pm:programms/Matlab/program.m,outputfile.txt,walltime=01:10:00
```

automatlab

Alias: `am`
Vstupné parametre:povinné parametre `program`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`,`output`.
Popis: Táto metóda vykonáva plne automatizované vytvorenie a zaradenie matlabovskej úlohy do fronty,tzn. vykonajú sa po sebe metódy `prepare_matlab` a `submit_job`.
Príklady použitia:
```
fab salomon automatlab:programms/Matlab/program.m,outputfile.txt
fab salomon am:programms/Matlab/program.m,outputfile.txt,walltime=01:10:00
```

prepare_job_array

Alias: `pja`
Vstupné parametre:povinné parametre `program`, `input_prefix`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`,`output`.
Popis: Metóda, ktorá vytvára PBS skript pre spústenie úloh typu Job Array (pole úloh). Princíp je rovnaký ako pri metóde `prepare_job`, navyše však obsahuje parameter `input_prefix`, čo značí názov vstupných súborov. Tieto súbory sa umiestnia do tzv. *task-listu*. Po vytvorení PBS skriptu sa celá zložka spolu so vstupnými súbormi presunie na vzdialené zariadenie.
Príklady použitia:
```
fab salomon prepare_job_array:programms/prog1.cpp,input
fab salomon pja:programms/prog1.cpp,input,walltime=01:10:00
```

[4] Ak súbor nevyžaduje kompiláciu, táto metóda sa preskočí

submit_job_array

Alias: `sja`
Vstupné parametre: povinné parametre `jobname`, `begin`, `end`, `pbs`, `output`
Popis: Táto metóda slúži na zaradenie úlohy typu Job array do príslušnej fronty a to tak, že sa spustí PBS skript zadaný v parametri `PBS`. Cesta k tomuto skriptu začína od adresára v premennej `env.remote_programms_path`. Hodnota parametra `jobname` môže byť ľubovolná. Parametre `begin` a `end` predstavujú ohraničenie vstupných súborov.
Príklad použitia:
```
fab salomon submit_job_array:myjob,1,100,software/my_pbs
```

autoarray

Alias: `aa`
Vstupné parametre: povinné parametre `program`, `input_prefix`, `jobname`, `begin`, `end`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`, `output`.
Popis: Táto metóda vykonáva plne automatizované zaradenie úloh typu Job array do príslušnej fronty. Vykonajú sa teda po sebe metódy `prepare_job_array` a `submit_job_array`. Ak je to nutné, vykoná sa aj kompilácia súboru.
Príklad použitia:
```
fab anselm aa:job_array/script.sh,file,pokus,1,101,cpus=16
```

prepare_gnu_parallel

Alias: `pgp`
Vstupné parametre:povinné parametre `program`, `input_prefix`, volitelné parametre `queue`, `nodes`, `cpus`, `walltime`, `omp`, `mpi`.
Popis: Metóda vytvára PBS skript pre úlohy s využitím technológie GNU parallel. Vstupný skript zadaný v parametri `program` je obalený príslušnými príkazmi, aby sa dal exekuovať pomocou GNU parallel. Cesta k tomuto skriptu pokračuje od adresára v premennej `env.remote_programms_path`. Opäť je možné do PBS skriptu začleniť načítanie užívateľom zvolených modulov.
Príklad použitia:
```
fab salomon prepare_gnu_parallel:myjob,software/my_pbs
```

submit_gnu_parallel

Alias: `sgp`
Vstupné parametre: povinné parametre `jobname`, `pbs`
Popis: Táto metóda vykonáva zaradenie úlohy do plánovača a to konkrétne pomocou PBS skriptu. Hodnota v parametri `jobname` môže byť ľubovolná. Cesta k PBS skriptu `pbs` začína od adresára `env.remote_proggrams_path`. **Príklady použitia**:
```
fab salomon submit_gnu_parallel:programms/Matlab/program.m,outputfile.txt
fab salomon sgp:programms/Matlab/program.m,outputfile.txt,walltime=01:10:00
```

7.3 Metódy pre prácu s Gitom

V rámci čo najlepšej integrácie so systémom riadenia revízií Git boli do FabSimu implementované základné metódy, ktoré s ním umožňujú plnohodnotnú prácu. Pre využívanie nasledujúcich metód je nutné mať v konfiguračných súboroch vyplnené položky `git` a `git_path`

clone

Vstupné parametre: adresa priečinka, do ktorého sa má repozitár naklonovať.
Popis: Metóda, ktorá zabezpečuje naklonovanie repozitára do príslušného adresára `folder`. Táto adresa je naväzuje na adresár v premennej `env.home_path`.
Príklady použitia:
`fab salomon clone:software/git_folder`

push

Vstupné parametre: súbory, na ktoré sa metóda aplikuje, správa do commitu
Popis: Táto metóda vykoná vzdialene príkaz `git push`, teda aktualizuje vybrané súbory v repzitári. Krátka správa, ktorá sa umiestni do commitu (commit message) musí byť vždy uvedená ako posledný parameter.
Príklady použitia:
`fab salomon push:programms/prog1.cpp`

pull

Vstupné parametre: žiadne
Popis: Táto metóda vykoná vzdialene príkaz `git pull`, teda stiahne súbory z repozitára a aktualizuje ich. **Príklad použitia**:
`fab salomon pull`

checkout

Vstupné parametre: Názov vetvy, na ktorú sa má metóda aplikovať.
Popis: Metóda vykoná vzdialene príkaz `git checkout nazov_vetve`, tzn. prepne sa na užívateľom zadanú vetvu repozitára.
Príklad použitia:
`fab anselm checkout:master`

new_branch

Alias: `nb`
Vstupné parametre: názov vetvy, ktorá sa má vytvoriť.
Popis: Táto metóda vykoná vzdialene príkazy `git checkout master` a `git checkout -b nazov_vetve`. Z vetvy `master` sa teda vytvorí nová vetva s názvom, ktorý zadal užívateľ.
Príklady použitia:
`fab salomon new_branch:production`

7.4 Riadiace a organizačné metódy

Riadiace a organizačné metódy sa nepodieľajú na kompilácii programov, generovaní PBS skriptov či plánovaní úloh, poskytujú však užitočné vlastnosti, ktoré prispievajú k jednoduchšej manipulácii a spravovaniu prostredia.

backup

Vstupné parametre: cesty k jednotlivým priečinkom, resp. súborom, prípadne samostatný parameter `all`.
Popis: Metóda, ktorá zálohuje vybrané priečinky, resp. súbory do adresára `env.backup_path`. Cesta k súborom pokračuje od priečinka v `env.remote_home_path`), teda pri zadaní parametra napr. `programms/prog1.cpp` sa FabSim pokúsi nájsť a skopírovať súbor `env.home_path/programms/prog1.cpp`. Pri použití parametra `all` sa zazálohujú všetky súbory a podpriečinky z priečinka `env.home_path`. Priečinok, do ktorého sa zálohuje, má názov v tvare časového razítka (timestamp) momentu, kedy sa táto metóda spustila.
Príklady použitia:
`fab salomon backup:programms/prog1.cpp`,
`fab anselm backup:all`.

restore

Vstupné parametre: verzia zálohy, cesty k súborom na obnovenie, voliteľný parameter `tolerate`
Popis: Táto metóda je opakom `backup`, teda umožňuje obnoviť zálohované súbory na clustri. Alternatívnym použitím môže byť jednoduchý prenos súborov medzi dvoma zariadeniami v kombinácii s metódou `backup`. Parameter, ktorý špecifikuje verziu zálohy, môže byť buď konkrétny názov priečinka v adresári `env.backup_path`, avšak môže mať aj tvar `oldest`, resp. `latest`. V takom prípade sa obnovia súbory zo zálohy, ktorá bola vykonaná ako prvá, resp. posledná[5]. Druhý parameter je cesta k súboru, ktorý môže mať znova, podobne ako pri metóde `backup`, názov `all`. V tomto prípade sa obnovia všetky súbory. Pri zadaní volitelného parametra `tolerate=True` dôjde k zachovaniu takých súborov, ktoré sa v zálohe nevyskytujú.
Príklad použitia:
`fab anselm restore:latest,all`,
`fab salomon restore:2016_04_19_03_12_16_599627,programms/program.cpp`,
`fab salomon restore:oldest,all,tolerate=True`

format

Vstupné parametre: žiadne
Popis: Metóda, ktorá zmaže všetky súbory a adresáre v priečinku `env.remote_home_path`. Ešte predtým je však užívateľ informovaný o tom, že nasledujúca operácia spôsobí stratu všetkých súborov a je vyzvaný pre potvrdenie svojho rozhodnutia. Následne je ešte vyzvaný na zálohovanie súborov, ktoré sa chystá zmazať. Až po tomto sa samotná operácia vykoná.
Príklad použitia: `fab anselm format`

[5] Toto samozrejme platí len pri zachovaní názvov priečinkov so zálohami, keďže rozpoznávanie pracuje na princípe rekonštrukcii časového razítka

check_env

Alias: `ce`
Vstupné parametre: žiadne
Popis: Po spustení tejto metódy dôjde ku komplexnej previerke prostredia. V prvom kroku program skontroluje, či sú v konfiguračných súboroch vyplnené všetky povinné položky. V prípade, že nie sú vyplnené všetky, vypíše sa zoznam chýbajúcich položiek a proces sa ukončí. Inak metóda pokračuje tak, že kontroluje, či príslušné adresáre na vzdialenom prístroji skutočne existujú. Neexistujúce adresáre sa automaticky vytvoria. V poslednom kroku sa spracujú položky z riadiacich premenných `aliases` a `modules`. Následne sa upraví súbor `.bashrc` na vzdialenom zariadení tak, aby obsahoval jednotlivé aliasy, resp. aby sa pri každom pripojení načítali príslušné moduly. Za zmienku stojí pripomenúť, že ak súbor `.bashrc` obsahuje predtým definované aliasy, resp moduly a následne sa z konfiguračného súboru odstránia, dôjde k ich vymazaniu aj v súbore `.bashrc`.
Príklad použitia: `fab anselm check_env fab anselm ce`

7.5 Ostatné metódy

find_module

Alias: `fm`
Vstupné parametre: žiadne
Popis: Metóda, ktorá slúži na vyhľadávanie modulov nachádzajúcich sa na vzdialenom zariadení. Vyhľadávanie je case - insensitive (netreba dodržiavať veľké, resp. malé písmená v názvoch)
Príklad použitia: `fab anselm find_module:gcc`

diagnose

Vstupné parametre: povinný parameter `job_nr`
Popis: Táto metóda poskytuje výpis detailných informácií o konkrétnej úlohe zaradenej vo fronte.
Príklad použitia: `fab anselm diagnose:1278558.dm2`

set_hdf5

Alias: `sh`
Vstupné parametre: žiadne
Popis: Metóda, ktorá stiahne a nainštauje knižnicu hdf5 na vzdialené zariadenie.
Príklad použitia: `fab anselm set_hdf5`

abort

Vstupné parametre: povinný parameter `job_nr`
Popis: Metóda, ktorá zruší zadanú úlohu ktorá je vo fronte. Užívateľ je upozornený na riziko spojené s touto metódou a musí svoje rozhodnutie potvrdiť. Následne je už operácia nezvratná.

Príklad použitia: `fab anselm abort:1278561[].dm2`

check_progress

Alias: `cp`
Vstupné parametre: povinný parameter `path`
Popis: Táto metóda umožňuje interaktívne sledovanie výstupných súborov ktoré produkujú simulované programy. V týchto súboroch sa často nachádza progres simulácie, odhadovaný koniec a pod. Táto metóda pracuje v nekonečnom cykle, dá sa ukončiť klávesovou skratkou `Ctrl + C`. Atribút `path` začína od adresára `env.remote_programms_path`.
Príklad použitia: `fab anselm check_progress:examples/output.txt fab anselm cp:examples/ou`

put_file

Alias: `pf`
Vstupné parametre: povinné parametre `local`, `remote`
Popis: Z názvu je jasné, že sa jedná o metódu, ktorá presunie súbor alebo adresár z priečinka na lokálnom počítači `local` do priečinka `remote` na vzdialenom zariadení. Ak je presúvaný súbor, resp. adresár vúčší ako 1 GB, užívateľ dostane možnosť rozdeliť súbor na niekoľko segmentov (LFS stripes). Tento postup je odprúčaný, znižuje sa tak možnosť zahltenia uzlu pri práci s veľkými súbormi. Hodnoty parametrov môžu byť aj symbolické, t.j. môžu obsahovať premenné `env` zadané v konfiguračnom súbore.
Príklad použitia:
`fab anselm pf:programms/example/example.cpp,env.remote_programms_path/example`

get_file

Alias: `gf`
Vstupné parametre: povinné parametre `remote`, `local`
Popis: Opak metódy `put_file`, prenos súborov sa deje smerom zo vzdialeného zariadenia na lokálne. V parametroch opäť môžu byť symbolické hodnoty.
Príklad použitia:
`fab anselm gf:env.remote_programms_path/example,programms/example/example.cpp`

7.6 Vybrané metódy z pôvodnej verzie FabSimu

Hoci tento manuál sa zaoberá metódami implementovanými v rámci rozšírenia FabSimu, je na mieste spomenúť niekoľko užitočných metód, ktoré sú už implementované a je odporúčané užívateľom ich používať.

stat

Vstupné parametre: žiadne
Popis: Metóda, ktorá vypíše všetky užívatelove úlohy, ktoré sa v danom momente vykonávajú alebo sú zaradené vo fronte a čakajú na exekúciu.
Príklad použitia:
`fab anselm stat`

monitor

Vstupné parametre: žiadne
Popis: Metóda, ktorá periodicky vypisuje stav plánovača, tzn. každé dve minúty zavolá metódu `stat`. Keďže táto metóda beží v nekonečnom cykle, jej ukončenie je možné pomocou klávesovej kombinácie `Ctrl + C`.
Príklad použitia:
```
fab anselm monitor
```

wait_complete

Vstupné parametre: žiadne
Popis: Metóda, ktorá „zablokuje“ terminál pre ďalšie použitie a skončí až vtedy, keď budú všetky aktuálne naplánované úlohy skončené. Aj túto metódu možno násilne ukončiť klávesovou skratkou `Ctrl + C`.
Príklad použitia:
```
fab anselm wait_complete
```

Kapitola 8

Zhodnotenie riešenia a plány do budúcna

Posledným krokom je objektívne zhodnotenie kvality riešenia, jeho predností ale i nedostatkov. Dosiahnuté riešenie treba porovnať s určenými cieľmi a v prípade ich nenaplnenia nájsť príčinu, prečo sa to nepodarilo. Na mieste je aj zamyslenie sa nad budúcim vývojom programu, jeho smerovanie a možnosti vylepšenia.

8.1 Zhodnotenie riešenia

Implementovaný modul obsahuje všetky funkcie ktoré boli stanovené ako kľúčové pri zbere informácií (viď kapitola 4.2). Je teda zabezpečené automatizované spúšťanie programov na superpočítačoch a generovanie skriptov potrebných na tento úkon. Zahrnuté sú aj menej populárne typy úloh ako napríklad polia úloh či GNU Parallel. Nad týmito úlohami je možná plná kontrola, či už získaním informácií o ich stave vo frontách, komplexných detailov či možnosťou celú úlohu z fronty odstrániť.

Pre zabezpečenie lepšej interakcie medzi užívateľom a programom bola vytvorená sada metód pre presun súborov či prepojenie so systémom pre správu revízií Git čím sa dosiahol sa lepší prístup k rôznym súborom, ako bol v predchádzajúcej verzií. Toto tvrdenie len umocňujú implementované metódy umožňujúce zálohovanie či obnovenie súborov na clustroch.

S dávkou sebakritiky však treba uznať, že ani toto riešenie nie je stopercentné. Program postráda napríklad možnosť vytvorenia vlastných modulov.Ďalším nepriaznivým aspektom je aj fakt, že testovanie na užívateľoch bolo možné začať až v dobe odovzdania projektu. Boli sme teda ochudobnení o časť spätnej väzby. Tento deficit bol však nahradený priebežnými hodnoteniami od vedúceho práce a aktívnejším testovaním počas vývoja.

8.2 Plány do budúcna

Je na mieste tvrdiť, že implementovaný modul poskytuje solídny základ pre plnohodnotnú obsluhu nielen superpočítačov Anselm a Salomon. Stále je však priestor pre vylepšenia. Tie sa budú uberať smerom k eliminácií vyššie uvedených nedostatkov, teda predovšetkým vytváraniu vlastných modulov. Očakávaná je aj spätná väzba od užívateľov, ktorá pomôže odhaliť a vyriešiť chyby, ktoré boli počas implementácie a testovania prehliadnuté.

Kapitola 9

Záver

Cieľom práce bolo rozšíriť existujúci program FabSim o modul umožňujúci plnohodnotnú obsluhu superpočítačov, špeciálne superpočítače Anselm a Salomon. Modul musel byť nenáročný na používanie, doplnený o praktický užívateľský manuál. Analýzou požiadavok sme zistili, ktoré vlastnosti, resp. metódy aktuálnej verzie FabSimu sú postačujúce a ktoré treba doplniť, aby bola zaistená plná funkcionalita podľa predstáv užívateľov. Štúdium koncových užívateľov programu nám zasa udalo smer vývoja, ktorým sa treba uberať, aby bol program dostatočne intuitívny a ľahko obslúžiteľný aj ľuďmi s malými skúsenosťami s konzolovými aplikáciami. Pri vývoji boli použité metódy zo softvérového inžinierstva, ktoré zaručili organizovaný postup vývoja.

Výsledný program spĺňa všetky požiadavky zadávateľa úlohy. Nad rámec týchto požiadavok boli pridané metódy ktoré umožňujú lepšiu integráciu so systémom riadenia revízií Git, metódy pre uľahčený prenos súborov či zálohovanie, resp. obnovenie súborov na clustri. Takisto, pre jednoduchšie ovládanie a prechádzanie neočakávanému chovaniu programu boli implementované kontrolné mechanizmy, ktoré upozorňujú užívatela napríklad na nedostatočne vyplnené konfiguračné súbory.

Program bol otestovaný nie len počas vývoja, ale aj po jeho ukončení. V dobe odovzdania sa začalo s testami na koncových užívateľoch. Napriek tomu je program funkčný, s plnohodnotným užívateľským manuálom a pripravený na použitie. Zároveň potvrdil svoju úlohu, ktorou je zjednodušenie práce a ušetrenie času pri obsluhe superpočítačov.

Ďalším smerovaním vývoja programu je implementácia vytvárania vlastných modulov a analýza spätnej väzby získanej testovaním na užívateľoch.

Literatúra

[1] Cadle, J.; Yeates, D.: *Project Management for Information Systems*. Pearson Education Limited, páté vydání, 2008, 131 s., iSBN 978-0-13-206858-1.

[2] The Cray-1 Computer System. online, 1977.
URL http://archive.computerhistory.org/resources/text/Cray/Cray.Cray1.1977.102638650.pdf

[3] Downey, A. B.: *Python for Software Design*. How to Think Like a Computer Scientist, Cambridge University Press, 2009, 1–2 s., iSBN 978-0-521-72596-5.

[4] Fernandez, M. R.: Nodes, Sockets, Cores and FLOPS, Oh, My. online, 2010-03-08 [cit. 2015-12-01].
URL http://en.community.dell.com/techcenter/high-performance-computing/w/wiki/2329

[5] Gupta, S.: China's investment in GPU supercomputing begins to pay off big time! online, 2009-06-09 [cit. 2016-05-12].
URL https://blogs.nvidia.com/blog/2011/06/09/chinas-investment-in-gpu-supercomputing-begins-to-pay-off-big-time

[6] Jenkins, N.: A Software Testing Primer. online, 2008.
URL http://www.nickjenkins.net/prose/testingPrimer.pdf

[7] Jirasek, P.: Introduction. online, 2015-09-30 [cit. 2016-05-13].
URL https://docs.it4i.cz/salomon

[8] Krauthammer, C.: Be Afraid. online, 1997-05-26 [cit. 2016-05-12].
URL http://www.weeklystandard.com/be-afraid/article/9802

[9] Křena, B.; Kočí, R.: Úvod do softwarového inženýrství. online, 12 2010 [cit. 2016-05-04].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IUS-IT/texts/IUS_opora.pdf?cid=8697

[10] Lopez, P. E.: hpcugent/easybuild-easyconfigs.
https://github.com/hpcugent/easybuild-easyconfigs/blob/develop/easybuild/easyconfigs/c/Chimera/Chimera-1.10-linux_x86_64.eb, 2016.

[11] Myers, G. J.; Badgett, T.; Sandler, C.: *The art of software testing*. John Wiley & Sons, Inc., 2012, 2 s., iSBN 978-1-118-13313-2.

[12] Rouse, M.: Definition supercomputer. online, 2008-09-01 [cit. 2015-12-01].
URL http://whatis.techtarget.com/definition/supercomputer

[13] Society, I. C.: *IEEE Standard for Software and System Test Documentation*. Institute of Electrical and Electronics Engineers, Inc, 2008, 11 s., iSBN 978-0-7381-5746-7.

[14] The BSD License Problem. online, 1998 [cit. 2016-04-28].
URL http://tech-insider.org/free-software/research/1998/0813.html

[15] Software Engineering Tutorial. online, 2014.
URL http://www.tutorialspoint.com/software_engineering/software_engineering_tutorial.pdf

[16] van der Vlist, E.: Evans moves against angle brackets in MinML. online, 2001-05-12 [cit. 2016-04-28].
URL http://www.xmlhack.com/read.php_item=1213

[17] Willmor, D.; Embury, S. M.: A safe regression test selection technique for database–driven applications. *Proceedings of the 21st IEEE International Conference on Software Maintenance*, 9 2005: str. 1.

[18] WWW stránky: TOP 500 List November 2015. online.
URL http://www.top500.org/lists/2015/11/

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúcu adresárovú štruktúru:

- `docs/` - technická správa a jej zdrojové kódy
- `src/` - adresár zdrojových súborov programu FabSim
- `install.sh` - inštalačný skript

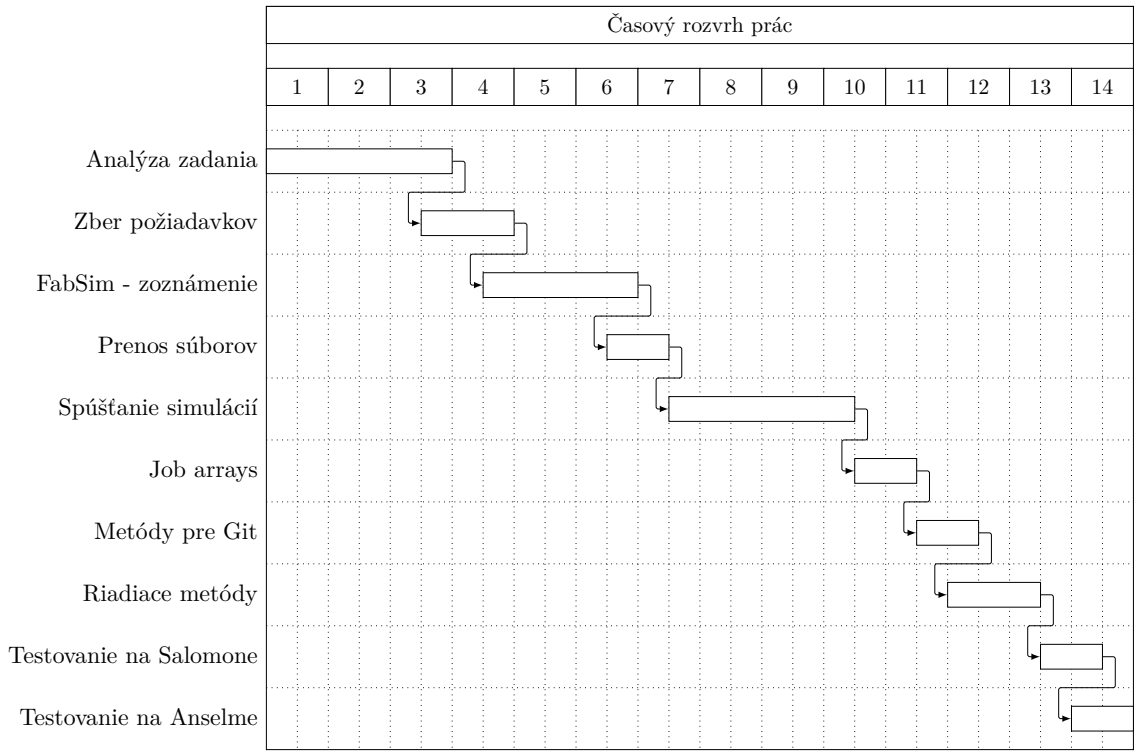
Zdrojové kódy programu FabSim sa nachádzajú v adresári `src/deploy`. Tento adresár obsahuje pôvodné aj nové zdrojové súbory. Súbory, ktoré patria do tejto práce sú `it4i.py` a `template_bodies.py`.

Za zmienku stojí aj adresár `src/deploy/programms`, ktorý obsahuje testovacie programy na overenie funkčnosti implementovaných riešení. Pred spustením je nutné vyplniť súbor `machines_user.yml`, aby bolo možné spustiť jednotlivé metódy. Príkazy FabSimu sa musia spúšťať z adresára `deploy`. Inštalačný skript bol otestovaný na systéme Xubuntu 14.04.

Príloha B

Obrázky

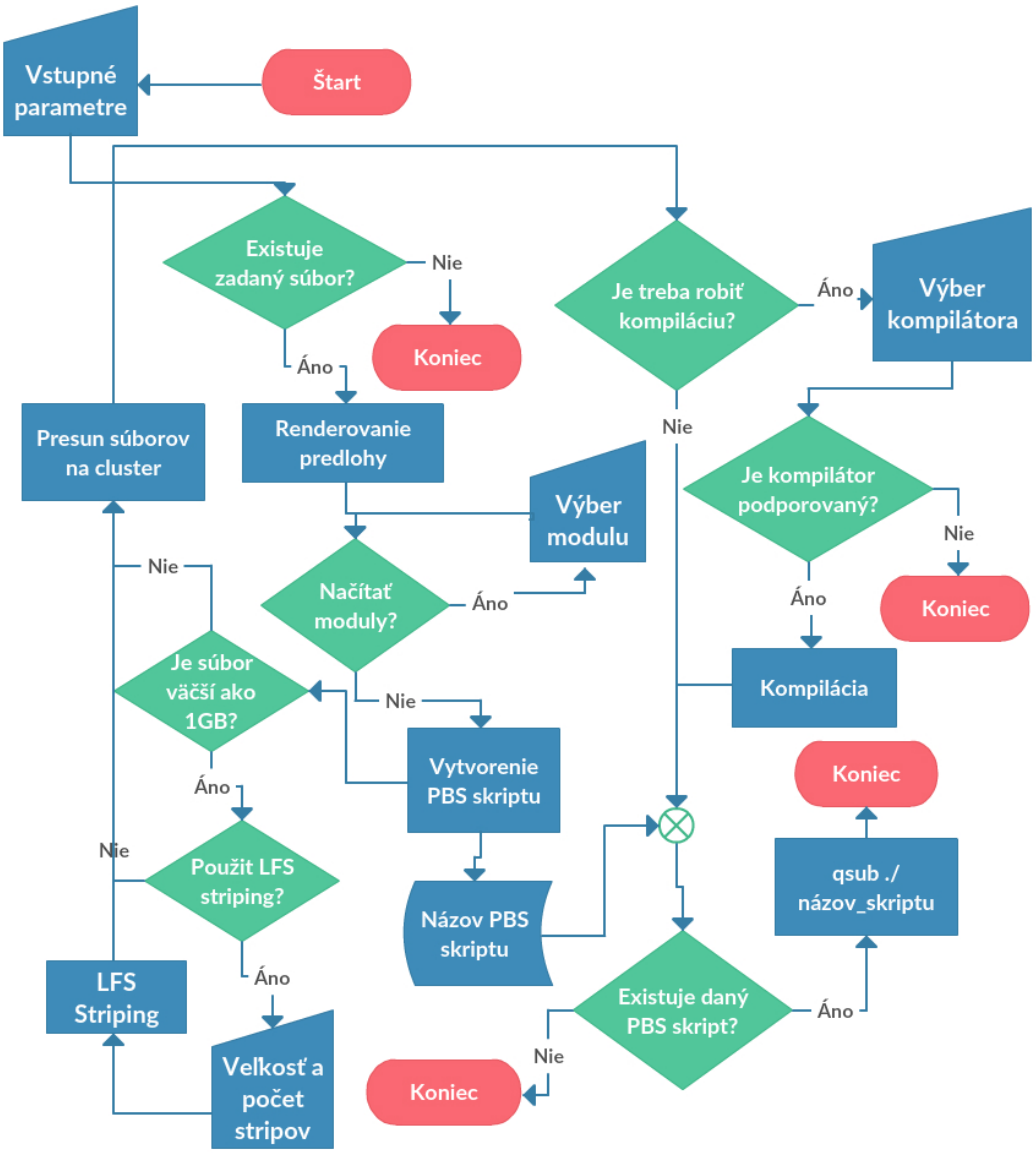
B.1 Ganttov diagram vývoja



Tento diagram zobrazuje priebeh postupu prác pri vývoji nového modulu do programu FabSim. Vývoj bol rozdelený na 14 týždňov, teda zhruba na dobu letného semestra. Práce v zimnom semestri tu neboli zahrnuté, pretože sa jednalo najmä o študovanie problematiky superpočítačov a iných teoretických aspektov. Práce takisto neboli pravidelné, preto by sa aj ťažko zaznamenávali do diagramu.

B.2 Diagramy prípadov použitia

B.2.1 Metóda autojob



Obr. B.1: Diagram prípadu užitia metódy `autojob`

Tento diagram prípadu užitia podrobne popisuje chovanie programu pri metóde `autojob`. Ako je možné vidieť z diagramu, zásadnú úlohu pri tejto metóde hrá aj užívateľ, nakoľko od jeho rozhodnutia závisí celý rad rozhodnutí (zvolený kompilátor, moduly, LFS striping a pod.).

Príloha C

Zdrojové kódy

C.1 EasyConfig súbor

```
# This file is an EasyBuild reciPY as per https://github.com/
    hpcugent/easybuild
# Author: Pablo Escobar Lopez
# Swiss Institute of Bioinformatics
# Biozentrum - University of Basel
name = "Chimera"
version = "1.10"
versionsuffix = "-linux_x86_64"
homepage = 'https://www.cgl.ucsf.edu/chimera/'

description = """ UCSF Chimera is a highly extensible program for
     interactive visualization
 and analysis of molecular structures and related data, including
      density maps, supramolecular
 assemblies, sequence alignments, docking results, trajectories,
     and conformational ensembles. """

toolchain = {'name': 'dummy', 'version': ''}

# no public download URL. Go to https://www.cgl.ucsf.edu/chimera/
    download.html
sources = ['%(namelower)s-%(version)s%(versionsuffix)s.bin']

# unzip is required to uncompress the provided .bin file
osdependencies = ['unzip']

sanity_check_paths = {'files': ["bin/chimera"],'dirs': []}
moduleclass = 'bio'
```

Listing C.1: Vzor easyconfig súboru

C.2 Ukážka konfiguračného súboru

```
default:
  local_results: "$localroot/results"
  local_templates_path: "$localroot/templates"
  local_programms_path: "$localroot/programms"
  backup_path: "/home/peter/Coding/FSfinal/FabSim-master/deploy/
     backup"
  local_home_path: "/home/peter/Coding/FSfinal/FabSim-master/
     deploy"
  username: "xstrec05"
  local_results: "$localroot/results"
  local_configs: ""

salomon:
  username: "xstrec05"
  project: "OPEN-5-13"
  job_dispatch: "qsub"
  remote: "salomon.it4i.cz"
  key_filename: "~/.ssh/id_rsa-xstrec05"
  remote_home_path: "/scratch/work/user/xstrec05"
  remote_programms_path: "$remote_home_path/programms"
  remote_software_path: "$remote_home_path/software"
  git: 'git@gitlab.com:StrecanskyP/FabSim_Test.git'
  git_path: '$remote_home_path/git'
  aliases: {"home":"cd /scratch/work/user/xstrec05"}
  modules: []
anselm:
  username: "xstrec05"
  project: "OPEN-5-13"
  job_dispatch: "qsub"
  remote: "anselm.it4i.cz"
  key_filename: "~/.ssh/id_rsa-xstrec05"
  remote_home_path: "/scratch/xstrec05"
  remote_programms_path: '$remote_home_path/programms'
  remote_software_path: '$remote_home_path/software'
  git: 'git@gitlab.com:StrecanskyP/FabSim_Test.git'
  git_path: '$remote_home_path/breke'
  aliases: {"home":"cd /scratch/xstrec05"}
  modules: ["gcc/4.9.0"]
```

Listing C.2: Ukážka konfiguračného súboru

Tento konfiguračný súbor je určený pre superpočítače Anselm a Salomon. Tretia položka `default` značí jednotlivé cesty k adresárom na lokálnom zariadení.

C.3 Ukážka PBS skriptu pre klasické úlohy

```
#!/bin/bash
#PBS -q qexp
#PBS -A OPEN-5-13
#PBS -l select=1:ncpus=24:mpiprocs=0:ompthreads=0
#PBS -l walltime=00:01:00
#PBS -m be
#PBS -N KW-L25

export OMP_PROC_BIND=true
export OMP_PLACES=cores

module load numpy/1.9.1-intel-2015b-Python-2.7.9

cd $PBS_O_WORKDIR
./example
```

Listing C.3: Ukážka vytvoreného PBS skriptu

Jednoduchý PBS skript pre klasické úlohy. Navyše bol načítaný model `numpy`. Jednotlivé vlastnosti úlohy možno pozorovať na začiatku súboru v riadkoch začínajúcich znakmi `#PBS`.